VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA
EKONOMICKÁ FAKULTA

KATEDRA SYSTÉMOVÉHO INŽENÝRSTVÍ

Návrh a implementace systému evidence zákazníků pro privátní podnikovou síť neziskové organizace

Design and Implementation of the Customer Records System for Virtual Private Network of the Non-profit Organization

Student: Bc. Marek Pasz

Vedoucí diplomové práce: Ing. Vítězslav Novák, Ph.D.

Ostrava 2015

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra systémového inženýrství

Zadání diplomové práce

Student: **Bc. Marek Pasz**

Studijní program: N6209 Systémové inženýrství a informatika

Studijní obor: 6209T025 Systémové inženýrství a informatika

Téma: Návrh a implementace systému evidence zákazníků pro privátní podnikovou síť neziskové organizace
Design and Implementation of the Customer Records System for Virtual Private Network of the Non-profit Organization

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska a metodiky systému evidence zákazníků
3. Analýza a návrh systému v neziskové organizaci
4. Realizace, testování a zhodnocení systému
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků diplomové práce Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

FOWLER, Martin. *Destilované UML.* Praha: Grada, 2009. 173 s. ISBN 978-80-247-2062-3.
WALLS, Craig. *Spring in action.* 2nd ed. Greenwich: Manning, 2008. 730 p. ISBN 19-339-8813-4.
BAUER, Christian and Gavin KING. *Java Persistence with Hibernate.* 2nd ed. Greenwich: Manning Publications, 2007. 408 p. ISBN 19-323-9488-5.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Vítězslav Novák, Ph.D.**

Datum zadání: 21.11.2014
Datum odevzdání: 25.04.2015

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * OSTRAVA * Ekonomická fakulta 1

doc. Ing. Jana Hančlová, CSc.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Rád bych touto cestou poděkoval mému vedoucímu diplomové práce Ing. Vítězslavu Novákovi, Ph.D. za jeho cenné připomínky, rady a vedení při zpracování diplomové práce. Taktéž děkuji i panu Radimu Kantorovi za průběžné konzultace, poskytnuté informace a materiály nutné pro vytvoření požadovaného řešení. Poděkování patří i mé rodině za trpělivost a podporu.

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracoval samostatně.

V Ostravě dne 8. 4. 2015 .....

.....

Bc. Marek Pasz

Obsah

1 Úvod

Webové aplikace, ale i celá oblast informačních technologií, jsou v dnešní době velmi aktuální jak už v podnikové sféře, tak i ve sféře veřejné správy nebo dalších sférách jako například v neziskových organizacích. Využívání informačních systémů přináší úsporu v nákladech díky zvýšení efektivnosti workflow, a tím snížení potřebného času pro vykonání potřebného úkonu a tedy zvýšení výkonu za člověkohodinu. Informační systémy na rozdíl od papírových systémů, či jim podobných, můžou navíc zvýšit motivaci pracovníků díky využívání moderních technologií.

Webové technologie a s tím spojené i webové aplikace jsou velkým přínosem v podnicích a organizacích. Díky webovým technologiím lze využívat výhod desktopových aplikací v kombinaci s výhodami webových prohlížečů. Aplikace fungující přes webový prohlížeč může nabídnout uživatelům víceuživatelskou přístupnost, přístup k systému přes prohlížeč z jakéhokoli osobního počítače (dostupnost), ale i jednotné úložiště dat.

Tvorba informačního systému jako webové aplikace v rámci diplomové práce byla zvolena z důvodu reálného uplatnění v praxi, získání a prohloubení znalostí, které mohou určitým způsobem zvýšit hodnotu na trhu práce, a taky proto, že je po programátorech webových aplikací vysoká poptávka na pracovním trhu. Mimo jiné je tvorba webových aplikací moderní aktivita, a díky tomu není problém najít informace a novinky z této oblasti jak v internetu, tak v odborných časopisech.

Tato diplomová práce se zabývá, jak už bylo výše zmíněno, tvorbou a implementací informačního systému, který by měl využívat webových technologií. Jde tak nejen o vývoj a programování čisté webové aplikace, ale návrh informačního systému jako celku. V rámci webové aplikace půjde o analýzu požadavků, návrh struktury databáze, naprogramování aplikace, důsledné zaměření na zabezpečení aplikace, její naplnění daty a otestování všech funkcí. Výsledným produktem bude navržený informační systém, webová aplikace, a tato práce smí sloužit jako dokumentace vývoje. Diplomová práce bude sloužit k popisu technologií, metodologií, tvorby, implementace a testování požadovaného systému.

Cílem diplomové práce je analýza, návrh, implementace a testování systému evidence zákazníků pro privátní podnikovou síť neziskové organizace včetně splnění hlavních a dílčích cílů zadaných zadavatelem.

2 Teoretická východiska a metodiky systému evidence zákazníků

2.1 Technologie

V následujících odstavcích jsou popsány technologie potřebné pro vytvoření webové aplikace. Pro aplikaci jsou zvoleny technologie jako Java, konkrétněji v kategorii Enterprise Edition s komponentovým frameworkem Spring, který má Model-View-Controller architekturu, a objektově-relačním mapováním databáze (Hibernate).

2.1.1 Java

Svět se stal velmi vědecky rozvinutým hlavně díky vzniku mikroprocesorů, které vedly k tvorbě různých zařízení a systémů. Mikroprocesory jsou používány v počítačích, televizích, ale dokonce i v mobilních telefonech a tabletech. Všechny tyto zařízení jsou velmi užitečné hlavně díky jejich možnosti komunikace. Při zvyšování možností a snižování zpracování informací lokálně síť roste velmi rychle kvůli stále vyšším nárokům na přenos informací v elektronických systémech. Právě Internet umožňuje posílat informace mezi různými typy zařízení umístěných na odlišných místech. Informace tak může být dostupná rychle a levně na celém světě. Infrastruktura nejrůznějších elektronických zařízení připojených k síti tak vyžaduje prostředí, pro které je programovací jazyk Java velmi vhodný.

Motto vytvořené firmou Sun Microsystems pro programovací jazyk Java je: „Napiš jednou a spusť kdekoli“ (angl. WORA). Program v Javě, jenž je napsán například pro operační systém Windows, musí být spustitelný i na ostatních operačních systémech (Linux apod.). Pro nejčastější použití se Java platformy rozdělují na části:

- Java ME (Java Micro Edition) pro vestavěné systémy a mobilní zařízení,
- Java SE (Java Standard Edition) pro stolní počítače, servery a další zařízení,
- Java EE (Java Enterprise Edition) hlavně pro aplikační servery.

Java technologii lze rozdělit na různé komponenty, které jsou zapojené do běhu programu. Pět důležitých komponent Java Standard Edition programu jsou samotný programovací jazyk Java, rozhraní pro programování aplikací (Java APIs), Java Virtual Machine (JVM), souborové formáty Java tříd a nástroje nutné pro zavádění, kompilaci a ladění aplikace. (Buyaa, 2009)

Programovací jazyk Java lze považovat za objektově orientovaný (obsahuje zásady OOP a s výjimkou primitivních datových typů jsou všechny ostatní typy referenční), jednoduchý (snadno pochopitelná syntaxe a vývoj programů), dostupný (velké množství knihoven) a jak už bylo dříve zmíněno multiplatformní (zdrojový kód je prvně přeložen na bajtkód, potom interpretován přes Java Virtual Machine, jenž postykuje možnost spuštění programu na různých operačních systémech).

Datové typy v programovacím jazyce Java jsou rozděleny na primitivní (byte, short, int, long, float, double, char, boolean) a refereční (třídy, rozhraní, pole, výčty). Jako primitivní datový typ už nepočítáme řetězec znaků, jelikož ten už je reprezentován jako objekt třídy `java.lang.String`. Mezi objektově orientované vlastnosti jazyka Java lze počítat zapouzdřenost, dědičnost a vícetvarost.

Objekt je instancí třídy a obsahuje vlastnosti a metody. Jako vlastnosti lze počítat proměnné, které popisují objekt. Metody popisují chování objektu a jeho modifikaci. Metody můžou být určené jako `final` (nelze metodu přetížit), `static` (metoda dostupná bez tvoření objektu), `abstract` (abstraktní metoda). Přístupová práva `public` – veřejná, `protected` – chráněná, `private` – soukromá a `friendly` - přátelská lze přiřadit třídám, metodám a proměnným. V rámci přetěžování metod lze každou metodu přetížit jinými parametry. Typ parametru metody následně dává najevo, jaká implementace má být použita. Aby byl kód přehlednější a čistější, je vhodné nepřetěžovat operátory. (Pecinovský, 2009)

V tomto jazyce je možné programovat mj. vícevláknové programy. Takovýto program obsahuje dvě nebo více části, které můžou být vykonávány současně. Každá takováto část je nazvána „vlákno“ a každé vlákno běží samostatně podle své zadané úlohy. Proto lze považovat vícevláknové programy jako určitou součást víceúlohového procesu.

Existují dva různé typy multitaskingu, a to jsou procesně založený a vláknově založený. Je důležité pochopit rozdíl mezi těmito dvěma typy. Pro mnoho lidí je procesně založený přístup známější oproti druhému přístupu. Proces je v tomto případě program, který je spouštěný. Proto procesně založený multitasking je vlastnost, která umožňuje např. počítači spouštět dva nebo více programů současně. To znamená, že lze v jednom čase mít spuštěný např. textový editor a video záznam. Tedy v tomto případě je program nejmenší spustitelná jednotka. Ve vláknově založeném multitaskingu je nejmenší spustitelnou jednotkou právě vlákno. To znamená, že jeden program může vykonávat dvě a více úloh současně. V praxi lze toto pozorovat například, pokud textový editor má za úkol vytisknout stránky a zároveň

v něm lze formátovat text, dokud jsou tyto úlohy vedeny jako dvě oddělená vlákna. Proto procesně založená varianta pracuje s větším měřítkem a vláknově založená má detailnější pohled na problematiku. V programovacím jazyce Java je možné ovlivnit vláknově založený multitasking, procesně založený už je ale nemožný.

V páté verzi programovacího jazyka Java se objevil velký milník v rámci zavedení generických typů a přidání nebo objasnění vícevláknového, konkurenčního modelu. Díky generikům je možno vytvořit třídy, rozhraní a metody, které budou pracovat s různými typy dat při neporušení typové integrity. Mnoho algoritmů je logicky stejných, i když pracují s různými typy dat. Generiky umožňují programování algoritmu, který dokáže pracovat nezávisle na datovém typu, ať už jde o celočíselný datový typ, řetězec znaků, objekt nebo vlákno.

Vícevláknové aplikace dokážou efektivně využívat plný potenciál výpočetní síly systému. Jednou z cest, jakými je tohoto dosaženo, je udržovat dobu nečinnosti na minimu. Toto je zvláště důležité pro interaktivní a síťové prostředí, se kterými Java pracuje, jelikož v takovýchto prostředích jsou častější momenty nečinnosti. Například přenos dat přes síť je o mnoho pomalejší, než když by tato data zpracovával počítač. Dokonce čtení a zápis na lokální souborové systémy je mnohem pomalejší, než čas, který by zabral procesor pro jejich zpracování. Navíc data na vstupu zadána uživatelem jsou samozřejmě zadána mnohem pomaleji, než kdyby byla zadána počítačem. Program v jednovláknovém prostředí musí čekat na dokončení jednoho z těchto kroků pro postup k dalšímu. Proto většinu času musí být program nečinný, jelikož čeká na vstupy. Vícevláknový přístup tyto problémy (dobu nečinnosti při čekání) částečně řeší, protože jiné vlákno může běžet, když to původní vlákno čeká. (Schild, 2014)

Mezi novinky šesté vydané verze vydané v prosinci 2006 lze řadit zavedení managed beanů, context a dependency injection, validaci beanů, rozhraní pro poskytování autentifikačních služeb apod. V červenci roku 2011 došlo k vydání sedmé verze Javy. Mezi hlavní uvedené novinky této verze je počítán „Projekt Mince“ (Project Coin), jenž obsahuje malé změny v jazyce Java, které ale nemění samotný Java Virtual Machine (JVM). Konkrétně se jedná o používání znakových řetězců v přepínačích, vkládání podtržítek do číselných literálů, využívání více než jednoho bloku „catch“, možnost začít používat blok „try-with-resources“, zdokonalené rozhraní v rámci používání diamantových operátorů a vylepšení v použití metod s proměnným počtem argumentů. (Reese, 2012)

V osmé verzi jazyka Java se poprvé objevuje „Projekt Lambda“, který přináší nové uzávěry v syntaxi, reference metod a výchozí metody rozhraní. Umožňuje mnoho možností a výhod funkcionálních jazyků, aniž by ztratila srozumitelnost a jednoduchost, kterou Java vždy přinášela. Kromě „Projektu Lambda“ Java 8 přináší nové rozhraní API pro datum a čas, ale taky proudy, javaskriptový Nashorn engine, který umožňuje vývojářům vložit Javaskriptový kód uvnitř aplikace, použití anotací při tvoření nové instance, přetypování datových typů a odstraňuje permanentní generování (PermGen) z virtuálního stroje HotSpot. (Davis, 2014)

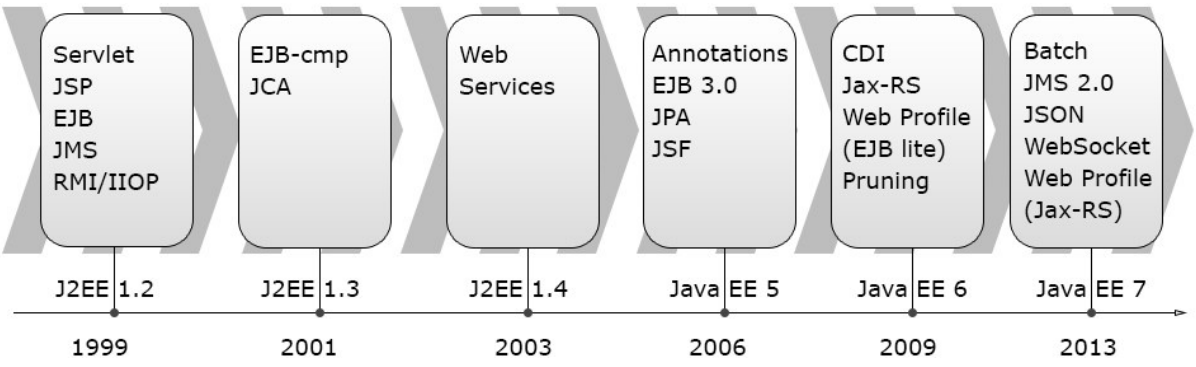
Lambda výrazy a jejich funkce významně ovlivňují programovací jazyk Java ze dvou hlavních důvodů. Za prvé jsou přidány nové elementy syntaxe, které zvyšují význam výrazů v tomto jazyce. Díky této vlastnosti je zjednodušená implementace některých společných konstruktorů. Přidáním lambda výrazů jsou za druhé vytvořené nové možnosti, které jsou začleněny do API knihovny. Mezi tyto nové možnosti lze počítat schopnost jednoduššího využívání výhod rovnoběžného zpracování dat ve vícejádrových prostředích, hlavně pokud spravuje operace typu „for-each“ nebo nových API proudů, které podporují datové operace konvenčních rour. Navíc je díky lambda výrazům umožněno využívat výchozí metody rozhraní a referenční metody, které poskytují odkazovat na metody bez jejího provedení.

Pro správné pochopení lambda výrazů je důležité pochopit samotný lambda výraz a funkční rozhraní. Lambda výraz je ve své podstatě anonymní metoda. Tato metoda je použita pro implementaci metody definované ve funkčním rozhraní. Výsledek lambda výrazu je anonymní třída. Funkční rozhraní je takové rozhraní, které obsahuje právě jednu abstraktní metodu. V běžném pojetí tato metoda znázorňuje význam tohoto rozhraní, a proto funkční rozhraní reprezentuje jednu činnost. Například rozhraní „Runnable“ je funkčním rozhraním, protože definuje pouze jednu metodu – `run()`. Proto tato metoda definuje význam rozhraní „Runnable“. Navíc funkční rozhraní definuje výsledný typ lambda výrazu. Důležitou poznámkou je, že lambda výraz může být použit pouze ve významu, v jakém je její výsledný typ specifikován.

Praktickým příkladem lambda výrazu může být: `() -> 123.45`. Tento lambda výraz se vyhodnotí jako konstanta. Výraz nepřijímá žádné parametry, a proto je seznam parametrů prázdný. Pouze vrací konstantu „123.45“. Z toho důvodu by se tento lambda výraz dal napsat podobně jako: `Double mojeMetoda() { return 123.45; }`. S tím rozdílem, že metoda definována lambda výrazem nemá svůj vlastní název. (Schild, 2014)

2.1.2 Java Enterprise Edition

Java Enterprise Edition (Java EE) vznikl na konci devadesátých let minulého století a přináší do jazyka Java platformu pro robustní software v podnikovém prostředí. První verze Enterprise Edition byla vytvořena pro distribuované komponenty. V dalších verzích přibývá podpora SOAP nebo webových služeb založených na REST architektuře. V té době je při vydání každé nové verze zpochybňována, špatně pochopená a použitá, velmi zkomplikovaná, soutěží s open source knihovnami a je viděna jako těžkopádní technologie. Z těchto a dalších výčitek a kritických komentářů se postupem času zdokonaluje, a v dnešní době je úsilí při vývoji této platformy soustředěno hlavně na jednoduchost, ale i vydatnost, přenositelnost a integritu. Při rychle vyvíjejících se technologiích není snadným úkolem pro tvůrce jazyka Java jej udržovat v jednoduchosti a vzájemné konzistenci, ale přesto a právě proto je úsilí cílené právě na tyto aspekty.



Obr. 2.1 Vývoj Java Enterprise Edition (Layka, 2014)

Při správě a manipulaci kolekce objektů většina programátorů nezačíná vývoj tvorbou hashovací tabulky. Na rozdíl od toho je použito rozhraní kolekce pro programování aplikací. Podobně je tomu tak i při programování webové aplikace, tj. transakční, zabezpečené, interoperabilní, distribuované aplikace není zvykem programovat své vlastní nízko-úrovňové rozhraní pro programování, ale použije se Java Enterprise Edition. Stejně tak jak Java Standard Edition poskytuje rozhraní pro správu kolekcí, Java EE poskytuje standardizovaný způsob, jak řešit transakce díky Java Transaction API (JTA), zprávy pomocí Java Message Service (JMS) nebo perzistenci s Java Persistence API (JPA). Proto lze o Javě EE konstatovat, že je to soubor specifikací pro vývoj podnikových aplikací. Jinak řečeno může být brána jako rozšíření platformy Java SE pro vývoj distribuovaných, robustních, výkonných a vysoce dostupných aplikací pro zkrácení délku vývoje. (Goncalves, 2013)

U platformy Java EE je využit zjednodušený programovací model. Nasazení XML popisovačů je volitelné. Místo toho lze vložit informace jako anotace přímo do zdrojového

kódu a Java EE aplikační server nakonfiguruje komponentu při nasazení a běhu programu. Tyto anotace jsou všeobecně používány z důvodu požadavku na vložení dat do softwaru, které by jinak byly odbaveny v nasazovaném popisovači. S anotacemi je možno do kódu vložit informace patřící ke konkrétní programové komponentě.

Dependency Injection (DI) nebo taky zřejmé předávání závislostí, případně vkládání závislostí, může být použito pro všechny zdroje, které jsou nutné pro programovou komponentu, přičemž efektivně skrývá v kódu dané aplikace tvorbu a vyhledání použitých zdrojů. Dependency Injection bývá používáno v Enterprise Java Bean (EJB) kontejnerech, webových kontejnerech a klientských aplikacích. Tato technika umožňuje Java EE kontejneru automaticky vkládat reference dalším požadovaným komponentám nebo zdrojům, při použití anotací.

Java EE je navržena, jak už bylo dříve zmíněno, pro podporu aplikací, které implementují podnikové služby týkající se zákazníků, zaměstnanců, dodavatelů, obchodních partnerů apod. Takové aplikace bývají už ve své podstatě komplikované a komplexní. Podniková data jsou většinou z různých zdrojů distribuována různým klientům.

Pro lepší kontrolu a správu těchto aplikací týkajících se různých uživatelů je zavedena střední vrstva. Vrstva reprezentuje prostředí, které je úzce spjato s odborem informačních technologií podniku. Tato prostřední vrstva běží typicky na dedikovaném serverovém hardwaru a má přístup ke všem službám podniku.

Aplikační model platformy Java Enterprise Edition definuje architekturu pro implementování služeb, jako jsou vícevrstvé aplikace, které přináší škálovatelnost, přístupnost a ovladatelnost potřebné právě podnikovými aplikacemi. Model rozděluje práci nutnou pro implementaci vícevrstvé služby na část:

- podnikové a prezentační logiky, která má být implementována vývojářem a
- standardní systémové služby poskytované Java Enterprise Edition platformou.

Při vývoji aplikací na Java EE platformě je použit model distribuovaných, vícevrstvých aplikací. Aplikační logika je rozdělena na komponenty podle jejich funkce a aplikační komponenty, které umožňují využívat tyto aplikace na různých elektronických přístrojích podle vrstvy v rámci vícevrstvého Java EE prostředí, ke kterému daná aplikace náleží.

Za příklady vrstev ve vícevrstvé aplikaci lze považovat klientskou vrstvu, která běží na klientském počítači (prohlížeč, webové stránky), dále webovou vrstvu, která je spuštěna na Java EE aplikačním serveru (JSP stránky) stejně jako řídící vrstvu (Enterprise nebo Spring beany, mj. i služby aplikace) a datovou, databázovou vrstvu běžící na databázovém serveru (spravovaného systémem řízení báze dat). I když webová aplikace může být tvořena všemi čtyřmi vrstvami, většinou jsou enterprise aplikace třívrstvé. Do vrstev se tak řadí klientská vrstva, řídící vrstva (zahrnující webovou vrstvu) a datová, či databázová vrstva. Tato architektura tedy rozšiřuje typickou dvouvrstvou architekturu klient-server, když mezi klientskou aplikaci a úložištěm dat přidává vícevláknový aplikační server. (Jendrock, 2010)

2.1.3 Spring Framework

Sun Microsystems vydala v prosinci roku 1996 specifikaci „JavaBeans 1.00-A“. Ta definovala softwarový model komponent pro jazyk Java. Přesněji definovala sadu zásad pro kódování, která umožňovala Java objektům být znovu použitelné a jejich jednoduché skládání do komplexnějších aplikací. I když byly původně Java Beany určeny jako obecné prostředky pro definování znovupoužitelných aplikačních komponent, byly používány především jako model pro budování uživatelského rozhraní. Vypadaly až moc jednoduše, resp. nevypadaly použitelné pro reálnou praxi.

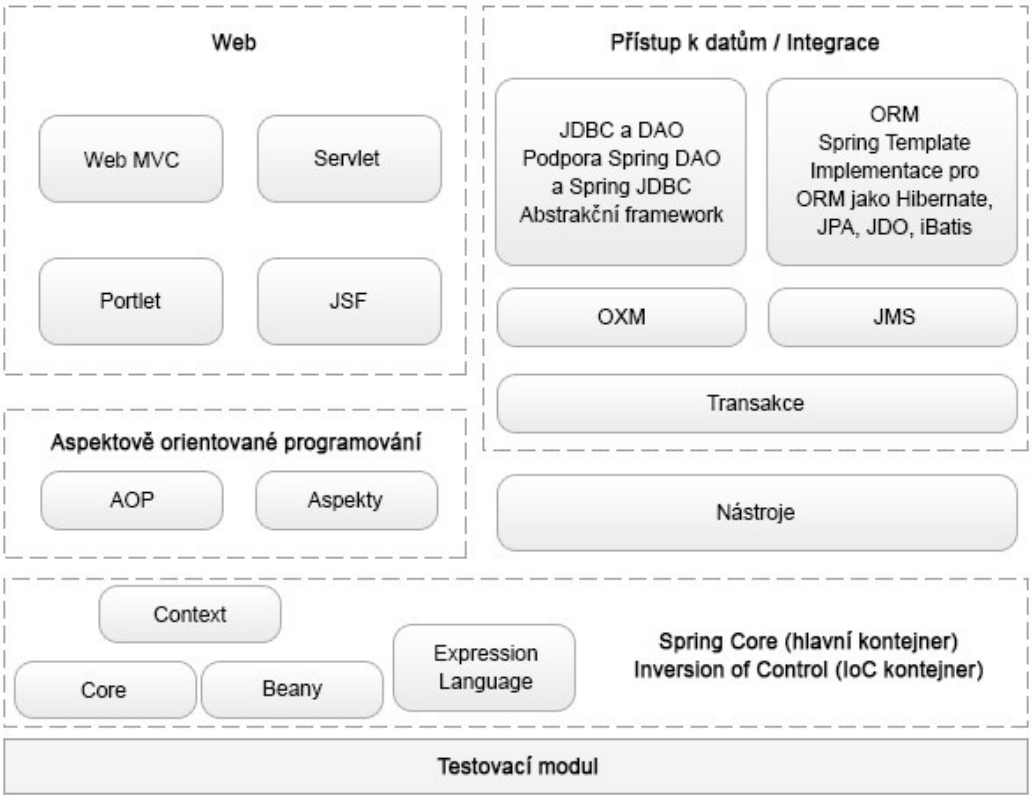
U komplexnějších aplikací je většinou nutná podpora transakcí, bezpečnosti a distribuované výpočetní síly, tj. služeb, které nejsou přímo poskytnuty Java Beany. Proto byly v březnu roku 1998 vyvinuty Enterprise Java Beany (EJB), které na jednu stranu zjednodušily vývoj aplikací, hlavně v oblasti transakcí a bezpečnosti, ale na druhou stranu vývoj značně zkomplikovaly tím, že přidaly povinné nasazení popisovačů a lokální nebo vzdálené rozhraní. V nynější době zahrnují programovací techniky aspektově-orientované programování (AOP) a dependency injection DI), které umožňují Java Beanům efektivitu, kterou přineslo EJB řešení, ale bez jeho složitosti. Nyní už není třeba psát těžkopádné komponenty založené na Enterprise Java Beanech, když samotné Java Beany bohatě stačí. Od první verze EJB se toho hodně změnilo a od třetí verze je samozřejmostí AOP nebo DI, a proto je jednodušší než předešlé verze, ale pro některé programátory přišla tato změna pozdě.

Mezi odlehčené frameworky lze počítat i Spring Framework. Je to open source aplikační rámec, který byl vytvořen Rodem Johnsonem. Jak už bylo řečeno dříve, byl vytvořen pro použití prostých Java Beanů k dosažení výsledků, které byly předtím možné

pouze při použití EJB. Výhody Spring Frameworku nejsou pouze na serverové straně, ale každá Java aplikace může využívat výhod Springu jako jednoduchosti, testovatelnosti a předávání závislostí.

Přesněji lze definovat Spring Framework jako lehký – kvůli své velikosti a režii, jelikož velikost samotného frameworku činí 2.5MB a velikost při jeho použití je zanedbatelná. Framework poskytuje předávání závislostí (reference) dalším komponentám a poskytuje taktéž velmi dobrou podporu aspektově orientovaného programování. Díky tomuto frameworku je umožněna správa životního cyklu a konfigurace objektů v aplikaci, a taky podporuje tvoření komplexních aplikací z menších komponent. (Walls, 2011)

Spring Framework se skládá z několika přesně definovaných modulů (viz Obr 2.1 Moduly Spring Frameworku). Při jejich užití jako celku, je skrze tyto moduly poskytnuto všechno, co je potřebné pro vývoj podnikových aplikací. Není nutné mít založenou aplikaci na celém Spring Frameworku. Je možno použít pouze některé moduly a využít i jiných možností, pokud tento samotný framework není dostačující. Navíc Spring poskytuje integraci s ostatními frameworky a knihovnami, takže není nutno psát znovu kód, který už je napsaný a připravený k použití.



Obr. 2.2 Moduly Spring Frameworku (Kumar, 2013 – volně přeloženo, upraveno)

Jak je uvedeno výše, všechny moduly jsou postaveny na hlavním (angl. core) kontejneru. Díky tomuto kontejneru je definována tvorba, konfigurace a správa beanů. Tyto třídy jsou implicitně využívány při konfiguraci aplikace. Pro vývojáře jsou ale zajímavější ostatní moduly, které využívají služeb poskytnutých hlavním kontejnerem. V další části jsou postupně blíže vysvětleny další moduly Spring Frameworku. (Walls, 2011)

a) Hlavní kontejner

Spring Core kontejner je základem Spring Frameworku. Poskytuje implementaci vzoru „Inversion of Control“ podporující vkládání závislostí. Zřejmé předávání závislostí je v praxi IoC kontejner implementovaný v jazyce Java pro propojení Java objektů. Hlavním úkolem Spring Core kontejneru je vytváření Java objektů vkládáním jejich závislostí v době vzniku a jejich správa. V tomto kontejneru lze vytvořit jakýkoliv Java objekt. Takový to objekt, který je vytvořen a spravován Spring Core kontejnerem se nazývá Spring Bean. To znamená, kterýkoli Java objekt se může stát Spring Beanem. Tímto je poskytnuto vhodné prostředí pro programování jednoduchých, ale isložitých podnikových aplikací, přičemž ve většině případů není potřeba propojovat (abstraktní) tovární třídy a metody. Taktéž se lze díky tomuto řešení vyhnout nutnosti programovat třídy s návrhovým vzorem singleton (unikáty).

Spring Core kontejner podporuje vkládání primitivních datových typů, polí, kolekcí ale i samotných Spring beanů. Vložení závislostí může proběhnout typicky přes konstruktor nebo setter metody. Anotační styl konfigurace vkládání závislostí zahrnuje podporu i pro jiné typy metod.

Konfigurace je pro tento kontejner velmi důležitá, protože je díky ní zprostředkována informace o objektech beanů, které musí tento kontejner inicializovat a spravovat jejich životní cyklus. Jednou z důležitějších vlastností hlavního kontejneru je, že nenutí vývojáři používat jeden formát konfiguračních souborů, a tedy lze používat formáty buď na bázi XML, souborových vlastností, anotací nebo programově při použití rozhraní Spring beanů. Třetí verze Spring Frameworku představuje Java beany založené na metadatech (JavaConfig) jako způsob konfigurace Spring beanů. Taky je možnost přizpůsobení Springu pro vlastní formáty konfiguračních souborů. Přesto většina vývojářů používá konfigurační soubory založené na XML nebo anotacích, jelikož jsou jednoduché a snadné k použití.

BeanFactory je implementací návrhového vzoru továrna (angl. factory). Skrze rozhraní BeanFactory je poskytnut základní koncový bod pro Spring Core kontejner

ve smyslu dosažení Spring beanů z kontejneru aplikace. Rozhraní ApplicationContext je podtypem BeanFactory a díky němu je poskytnuta dodatečná funkcionalita jako například správa posílaných zpráv v aplikaci, události, snadná integrace s funkcemi aspektově orientovaného programování a vymezení konkrétního rámce aplikační vrstvy. Pomocí tohoto rozhraní je podpořena celková infrastruktura a konkrétně jsou díky němu dostupné např. transakce.

Samotná inicializace Spring Core kontejneru je stejně snadná jako tvoření Java objektu. Inicializace je provedena tak, že je vytvořen BeanFactory nebo ApplicationContext objekt. Díky metodám rozhraní BeanFactory je umožněno získat instanci beanu, zjistit jeho existenci a typ nebo zjistit jestli má bean rozsah singleton či prototype. (Kumar, 2013)

b) Přístup k datům a integrace

Při práci s Java Database Connectivity (JDBC) vzniká často opakující se kód např. při získávání spojení s databází, vytváření dotazu, zpracování záznamů z tabulky a uzavírání spojení. Díky JDBC v rámci Springu a modulu Data Access Object (DAO) je umožňeno abstrahovat od znovu opakujícího se kódu a psát čistý a jednoduchý kód při práci s databází a předejít tak problémům, které můžou vést od běžné chyby po ztrátu spojení s úložištěm. Tento modul (DAO) buduje vrstvu smysluplných vyjímek nad vrstvou chybových hlášení vyhozených samotnými SŘBD, takže pro vývojáře už není nutno přesně chápat často nepřehledné a špatně čitelné SQL chybové hlášení. Kromě výše zmíněných vlastností, je modul DAO používán modulem AOP pro poskytnutí správy transakcí objektům vytvořeným ve Spring aplikaci.

Pro vývojáře, kteří preferují nástroje objektově-relačního mapování (ORM) nad čistým JDBC, je díky Spring Frameworku zprostředkován modul ORM. Tento modul v rámci Springu je postaven na modulu DAO a jeho prostřednictvím je docíleno snadného vývoje různých DAO nad několika ORM. V rámci Spring Frameworku není snaha o implementování vlastního řešení ORM, ale umožňuje integraci s několika populárními ORM aplikačními rámci. Mezi ně lze řadit Hibernate, Java Persistence API, Java Data Objects a iBatis SQL mapy. Pomocí správy transakcí ve Spring Frameworku je dovoleno využívat všechny výše zmíněné ORM aplikační rámce stejně jako JDBC. (Walls, 2011)

c) Transakce a anotace

Transakce jsou jedním z nejdůležitějších částí při budování podnikových aplikací. Nejběžnějším typem transakcí jsou databázové operace. Obvykle při pokusu o změnu

záznamů je spuštěna transakce, data jsou změněna a transakce je provedena, a posléze odvolána podle výsledků databázové operace. Nicméně se v mnoha případech častokrát stává, že požadavky aplikace a zdroje, se kterými musí aplikace komunikovat, jako např. ŠRBD, middleware pro na hlášení zpráv, ERP systém apod. způsobují, že správa transakcí je mnohem komplikovanější.

Správa transakcí by neměla být psána v rámci business logiky. Nejvhodnějším způsobem jak řešit správu transakcí je dovolit vývojářům definovat si vlastní transakční požadavky deklarativním způsobem a zpřístupnit možnosti jako Spring, Java EE nebo AOP pro zpracování transakční logiky právě uvnitř nich. Spring Framework poskytuje jak deklarativní, tak programovou správu transakcí.

Spring poskytuje vynikající podporu deklarativních transakcí, a proto vývojáři nemusí spravovat transakce uvnitř business logiky a tak nezpřehledňovat kód. Všechno co musí programátor udělat je deklarovat tyto metody, které potřebují transakční zpracování spolu s informacemi o konfiguraci transakcí a Spring se už postará o samotnou správu transakcí.

Pro sledování konkrétního chování kódu, který musí být spravován transakcemi, je vhodné využít ladění aplikace. Konkrétně se dá využít modul log4j, který umožnuje výpis transakčního zpracování na konzoli, či případě do textového souboru. (Ho a Harrop, 2012)

Při Spring anotacemi řízené aplikaci, jsou to právě anotace, které přiřazují transakce metodám, které potřebují transakční zpracování. Vlastnosti transakcí v aplikaci jsou propagace, izolace, zdali je transakce jen pro čtení a výjimky. Vlastnost jen pro čtení a výjimky jsou volitelnými atributy.

Propagace může být zvolená jako „Povinná“ (Required), což znamená, že kód vždycky proběhne jako transakce. Buďto vytvoří novou nebo použije už vytvořenou. Tato volba se liší s druhou možností „Povinně nová“ (Required new), kdy je transakce vždycky vytvořená nová, a pokud už existuje, pak se přeruší a vytvoří nová.

Druhá vlastnost Spring anotacemi řízených transakcí je jejich izolace. V aplikaci je izolace nastavená na výchozí hodnotu, to znamená, že se využívá izolace podle zvoleného systému řízení báze dat.[1] Tato aplikace používá SŘBD Oracle, a proto je izolace ve výchozím stavu nastavená na „Read commited“. Nižší úroveň „Read uncommited“, která umožňuje tzv.

[1] Více informací včetně zdrojového kódu izolační třídy: http://docs.spring.io/spring/docs/3.1.x/javadoc-api/org/springframework/transaction/annotation/Isolation.html#DEFAULT

„Dirty reads“ není v SŘBD Oracle možný, pouze vyšší úrovně izolace. „Dirty reads“ umožňují číst změněná data před zavoláním příkazu commit. Tato možnost není doporučována z důvodu, že nemusí proběhnout příkaz commit, ale příkaz rollback a tím pádem může uživatel aplikace vidět data, které neodpovídají realitě. Tato možnost, spolu s „povinnou“ propagací (Required propagation) tvoří dvojici pro stabilní používání databáze a jejích dat. Dalšími vlastnostmi transakcí jsou výjimky, které mají být odchyceny a vlastnost transakce jen pro čtení. (Walls, 2011)

V Javě Enterprise Edition šesté verze je poskytnuta sbírka anotací (JSR-330) pro vyjádření konfigurace dané aplikace v Java EE kontejneru nebo jiném kompatibilním IoC kontejneru. Spring Framework taktéž tyto anotace podporuje a rozpoznává, takže i když vývojář netvoří svou aplikaci v Java EE kontejneru, tak přesto může tyto anotace uvnitř Springu využívat. Díky těmto anotacím je umožněno ulehčit případnou migraci z původního Springového prostředí do Java EE 6 kontejneru nebo jiného kompatibilního IoC kontejneru (jako například Google Guice).

V případě, že vývojář chce využít sady těchto speciálních anotací, je nutno vložit potřebnou knihovnu do projektu. V tomto případě je to knihovna `javax.inject`, která je JSR-330 standardem. Po naimportování knihovny je možno využívat anotací. Jednou z těchto anotací může být `@Named`, jež deklaruje vložitelný bean (stejně jako `@Component` nebo `@Service` ve Springu). Dále je možno využít konstruktoru pro vkládání beanů `@Inject` apod. (Ho a Harrop, 2012)

d) Spring Web Flow

Spring Web Flow je framework díky kterému je umožněno tvořit tzv. „konverzační“ webové aplikace. Toto označení znamená, že uživatel používá aplikaci přirozeným způsobem. Aplikací jsou žádány nějaké informace a po jejich zadání uživatelem můžou být poslány zpět aplikaci, která tyto data zpracuje. Ve většině případů žádá aplikace větší množství informací.

Jako vhodný příklad lze uvést typické rozhraní aplikace jako průvodce (např. průvodce instalace softwaru). Většinou, v aplikacích sloužících jako průvodce je obsaženo několik stránek, které jsou zobrazeny jedna po druhé. Lze zadat nějaké informace a pak postoupit k další obrazovce, na které je opět možno zadat nějaké informace. Když si uživatel takovéto aplikace myslí, že některé informace zadal špatně, může se vrátit o krok zpět. Ale i když je uživateli umožněno se vrátit, vždy musí používat aplikaci takovým způsobem, jakým to její

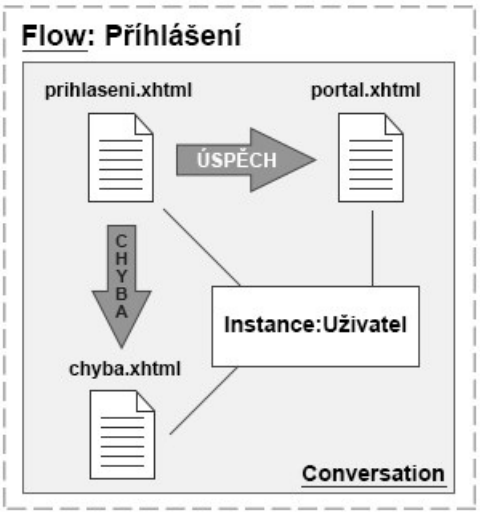
autor zamýšlel. Takový uživatel pak pracuje v předdefinovaném toku (angl. flow) s určitým cílem jako třeba objednání knížky nebo registrace nového uživatelského účtu.

I když je možné psát aplikace s takovýmto chováním i s jinými technologiemi, tak pomocí Spring Web Flow je umožněno vytváření programových toků velmi snadno. Takové toky pak nejenže nejsou oddělené od aplikační logiky, ale je taky možnost je znovu použít v jiných aplikacích. Programový tok ve Spring Web Flow frameworku je posloupnost takových kroků, ve kterých jsou obsaženy jejich stavy a přechody mezi nimi. Existují taktéž akce (actions), které lze provést v různých bodech programového toku, jako například když programový tok začíná nebo když je vykreslená webová stránka. Tok tedy vede k dosažení cíle, a proto lze tento modul využít pro různé webové aplikace. (Lüppken a Stäuble, 2009)

Při práci s programovými toky je pomocí Spring Web Flow postaráno o ovládání navigace na webových stránkách. Díky tomu je navigační logika přenesena z řídící logiky právě do programového toku, proto je taktéž možné spouštět činnosti přímo v programových tocích.

Webové toky tedy spravují navigaci na stránkách a v praxi lze s nimi např. spouštět metody servisní třídy přes výrazový jazyk, čímž skoro eliminují využití řadiče v řídící logice. Ovšem občasné běžné třídy v Javě jsou lepším řešením z toho důvodu, že v případě použití dlouhého a složitého výrazového jazyka je běžná Java třída mnohem čitelnější a přehlednější. Samozřejmě v případě jednoduchých metod s pár řádky v Javě postačí bohatě programový tok. Pokud jsou ale zapotřebí komplexnější metody s více řádky kódu, je vhodnější použít běžné Java třídy.

Jednou z hlavních výhod aplikačního rámce Web Flow je správa proměnných a stanovení jejich rozsahu. Podle specifikace servletu je jsou dostupné tři rozsahy platnosti proměnných (angl. scope of a variables). Jsou to rozsahy application, session a request. Proměnná v rozsahu application poskytuje proměnnou v rámci celého životního cyklu aplikace, proměnná session je dostupná tak dlouho, dokud ji vývojář neukončí nebo nevyprší její platnost a proměnná v rozsahu request je přístupná v rámci pouze jednoho požadavku. (Deinum a kol., 2012)



Obr. 2.3 Ukázka webového toku (Lüppken a Stäuble, 2009 – volně přeloženo)

Ve Web Flow jsou důležité tři elementy: flow, view a conversation. Dříve zmíněný webový tok (angl. flow) lze chápat jako podnikový proces reprezentující reálné procesy. Tok typicky obsahuje pohledy a data. Z technického hlediska představuje tok znovupoužitelné sled konkrétních kroků, který může být použit i v jiném kontextu. Pohled (angl. view) je představován jednou webovou stránkou, která zobrazuje informace vyžádané uživatelem. Element conversation lze chápat jako prvek slučující pohledy, které mezi sebou určitým způsobem komunikují nebo jsou jinak propojeny (viz Obr 2.2). (Lüppken a Stäuble, 2009)

Na rozdíl od standardní specifikace servetu, kde se vyskytují pouze tři rozsahy platnosti proměnných – application, session a request, poskytuje modul Spring Web Flow až pět rozsahů platností. Mezi ně lze řadit conversation, flow, view, flash a request.

Rozsah typu request je nejkratším ze všech a představuje rozsah mezi dvěma http požadavky. Při vytvoření http požadavku vzniká i tento rozsah a během něho může proběhnout nějaká událost. Po skončení požadavku končí i možnost využít proměnné z tohoto rozsahu. Rozsah flash je podobný do rozsahu request, ale ten je rozšířen o další požadavek request. Proto se v rozsahu scope vyskytují dva požadavky request. Jeden je požadavek na přesměrování a druhý je standardním GET požadavkem. Ukládat proměnné do tohoto rozsahu je vhodné tehdy, pokud je jich potřeba v pohledu, ale nikde jinde. View rozsah je přidělen při vstupu do pohledu a zničen při výstupu. Rozsah flow působí v daném toku nebo dílčím toku. Rozsah converstation je vhodný tehdy, když je potřeba přistupovat k objektu v rámci všech toků spojených mezi sebou navzájem. Je to rozsah, který lze popsat jako mezi stupeň rozsahu request a session, přičemž request je moc malý a session moc velký rozsah. (Deinum a kol., 2012)

e) Spring Security

Modul Spring Security je Java knihovna sloužící pro autentizaci a autorizaci. Lze jej použít v prostřední Java Enterprise Edition, Java servletech, ale i běžných aplikacích. V tomto modulu je obsaženo řešení pro většinu autentizačních/autorizačních problémů s čímž odstraňuje opakující se kód pro každou aplikaci.

V roce 2003 byl započat vývoj projektu Acegi Security a další rok byl vydán. V té době tento projekt poskytoval kvalitní řešení zabezpečení aplikací a mnoho aplikací založených na Spring Frameworku využívalo právě tohoto projektu. Postupem času, se zvětšující se popularitou Acegi a Springu byla tato kombinace velmi běžná. Spojení těchto projektů proběhlo v roce 2008, když byl Acegi začleněn jako oficiální dílčí projekt Spring Frameworku a pojmenován Spring Security.

Samotný jazyk Java je velmi komplexní jazyk a většinu úkolů, které má tento jazyk plnit je už připravena v různých knihovnách. Spring Security je tak důležitý framework, že je jedno jaký webový framework si vývojář ve výsledku zvolí, je možné Spring Security použít pro zabezpečení aplikace. Díky společnosti Oracle jsou dostupné návody Java EE Security pro webové i podnikové aplikace, jelikož jsou součástí Java EE kontejneru.

Použití Spring Security v projektu je vhodné, pokud je projekt založen na Spring Frameworku, pokud má projekt vysoké a komplexní požadavky na zabezpečení nebo pokud je potřeba integrace s jinými řešeními (protokoly) jako například OAuth, OpenID, LDAP apod. (Jagielski a Nabrdalik, 2013)

V rámci tohoto modulu lze zabezpečit aplikaci na různých úrovních. Konkrétně lze zabezpečit například webové adresy, pohledy, servisní metody a doménové modely. Z těchto úrovní lze vybrat, které budou zabezpečeny nebo případně jednotlivá zabezpečení kombinovat. Je tak možno zabezpečovat různé rozhraní, jako RMI, JMS a další.

Jelikož je Spring Security částí SpringSource portfolia, proto musí být veden jako open source softwarový nástroj. V rámci tohoto projektu existuje velká komunita, která se podílí na jeho vývoji a testování. Podílet se na vývoji open source projektů je lákává zkušenost pro mnoho vývojářů, protože díky tomu můžou vidět, jak komponenty fungují uvnitř, a může tyto komponenty zdokonalovat. (Scarioni, 2013)

2.1.4 Java Persistent API

Java Persistent API (JPA) je lehký, objektově založený framework pro perzistenci dat v programovacím jazyce Java. I když je hlavní komponentou API objektově relační mapování, tak tento framework poskytuje rovněž řešení pro architektonické výzvy při integraci s perzistentní vrstvou v podnikových aplikacích.

Jedním z hlavních cílů JPA je být jednoduchý na pochopení a snadný pro použití. I když problémy při návrhu doménového modelu nelze trivializovat, tak není řečeno, že technologie, která slouží vývojářům pro vypořádání se s těmito problémy, nesmí být jednoduchá a intuitivní.

Databázová entita, je samostatná jednotka, která může být vázána na neomezený počet dalších entit. V objektově orientovaném paradigma lze přidat takovéto jednotce chování a nazvat ji jako objekt. V JPA standardu může být objekt definovaný v aplikaci změněn na entitu.

Předtím, než je entita uložena do databáze, musí být zavolán Entity Managerem. Nejde jen o ukládání, ale jakékoliv změny v databázi. Právě tato třída je rozhraním, které téměř úplně zapouzdřuje běžnou práci s entitami. Pokud není právě Entity Manager použit pro vytvoření, čtení nebo zápis entity, není entita ničím víc než běžným nepersistentním Java objektem.

Když třída Entity Manager získá odkaz na objekt určité entity, a to už buď explicitním vložením jako argument metody nebo čtením z databáze, pak je daný objekt spravován právě třídou Entity Managera. Množina spravovaných instancí uvnitř entitního manažera v daném čase se nazývá perzistentní kontext (angl. persistance context). Pouze jedna instance Java objektu může existovat v persistentním kontextu. Například pokud instance třídy „Zaměstnanec“ s identifikátorem „128“ existuje v persistentním kontextu aplikace, pak žádná jiná instance se stejným identifikátorem v tomto kontextu nemůže existovat.

Entity Manager je nastaven tak, aby mohl ukládat nebo spravovat různé typy objektů, číst a zapisovat do databází a být implementován konkrétním poskytovatelem. Je to právě poskytovatel, který podporuje implementační engine pro celý Java Persistance API, od entitního manažera přes implementaci tříd s dotazy po samotné generování SQL jazyka. (Keith a Schincariol, 2013)

2.1.5 Hibernate

Hibernate je jedním z prvních open source objektově orientovaných frameworků, ve kterém je poskytnuto řešení na podnikové úrovni pro vývoj perzistentní vrstvy. Zavádění aplikace je ještě jednodušší při využití vlastností Spring Frameworku jako XML konfigurační soubory a zřejmé předávaní závislostí díky anotacím v Javě.

U objektově relačních frameworků je poskytnuta abstrakční vrstva nad užívanou perzistentní technologií (nejčastěji relační databází) díky níž je umožněno vývojářům soustředit se na objektově-orientované detaily jejich doménového modelu, než na věci týkající se přímo databáze. Existuje zde vnořený impedanční nesoulad mezi databázovými relačními tabulkami a objektově orientovanými třídami v Javě, což způsobuje potíže při implementaci efektivní abstrakce u objektově relačního mapování. Tento nesoulad je způsoben základními rozdíly mezi relačními databázemi a objektově-orientovanými jazyky, jako je třeba Java. Například v relačních databázích se nelze setkat s vlastnostmi OO jazyků jako polymorfismus, přístupnost nebo zapouzdření. Navíc pojem rovnosti mezi jazykem Java a SQL téměř neexistuje. Hibernate je jedním z řešení, jak snížit mezeru díky poskytnutí silného open source aplikačního rámce, jež vyjadřuje objektově orientovaný doménový model a ukazuje řešení, kdy tabulky a sloupce z databáze jsou synchronizovány s instancemi a vlastnostmi objektů v Java Beanech. (Bauer a King, 2007)

Navzdory zvýšení efektivity, díky které může být lépe vyvíjená perzistentní vrstva, může být integrace Hibernate do aplikace stále velmi složitý proces. Bez standardizované integrace byli vývojáři nuceni stále poznávat stejné poznatky o vývoji kódu a udržení infrastruktury potřebné pro použití Hibernate v jejich aplikacích, přičemž trávili hodně času a zdrojů. S postupem času, když byl Hibernate více a více populárnější se stával oblíbený i Spring Framework. Cílem Springu bylo usnadnit vývoj Java aplikací na straně serveru. Frameworky Spring a Hibernate jsou velmi úspěšné hlavně proto, že je skrze ně poskytnuto racionální a efektivní řešení než kdy mohla poskytnout technologie Enterprise Java Beanů. Jetoho docíleno tím, že Spring poskytuje jednoduchý deklarativní přístup ke správě transakcí a Hibernate poskytuje velmi robustní a intuitivní abstrakci objektově relačního mapování. Od dob úspěchu Springu a Hibernate byly více programovány aplikace, které byly jednodušší a méně náročné, čímž zvyšovaly snadnost jejich udržování a rozšiřitelnost.

Hibernate třetí verze neznamená to samé jako Enterprise JavaBeans. Díky Enterprise JavaBeans třetí verze je poskytnuta distribuovaná, zásobníková, serverová architektura.

Pomocí této technologie je poskytnuta specifikace např. pro distribuovanou správu transakcí, využití vláken, zpráv, webových služeb a bezpečnosti. V EJB třetí verze je předpokládáno, že správa databáze je záležitostí JPA.

Technologie Hibernate není ani Java Persistence API. Je to jedna z mnoha knihoven, díky níž je poskytnuta implementace JPA. První verze Hibernate, jež implementovala JPA 1.0 je verze 3.2, která byla zveřejněná na podzim roku 2006. Navíc Hibernate je ne jenom implementací JPA, ale i jejím rozšířením. Pomocí samotného jádra Hibernate je umožněno využívat jeho výhod bez použití jakékoli části JPA specifikace. Na druhou stranu, při použití pouze určitých části Hibernate, je umožněno jejich využití pouze díky JPA specifikaci. Silné propojení s JPA specifikací umožňuje přenositelnost ostatní implementace Java Persistence API jako třeba projekt Apache OpenJPA. (Fisher a Murphy, 2010)

Hibernate Query Language (HQL) je objektově orientovaný dotazovací jazyk pro získávání perzistentních objektů podle primárního klíče. Jak už bylo zmíněno dříve, v Hibernate jsou používány metody `get()` a `load()` pro získání perzistentních objektů podle jejich primárních klíčů. Proto je toto získání perzistentního objektu podle primárního klíče základním přístupem, ale v mnoha případech je buďto primární klíč neznámý, anebo je potřeba získat více perzistentních objektů jako výsledek jednoho nebo více parametrů hledání. Navíc, v některých případech je požadavek získat pár vlastností persistentního objektu nebo objektů namísto načítání kompletních objektů do paměti. V těchto případech je vhodné využít dotazy. Neznámější dotazovací jazyk je SQL. Lze proto využít jazyka SQL pro vytvoření dotazu, díky němuž je umožněno získání persistentních objektů mapovaných na tabulky relační databáze, což deleguje odpovědnost na ORM pro mapování JDBC ResultSetů na perzistentní objekty, ale pouze při použití SQL jazyka s nedostatky jako nepřenositelnost SQL dotazů mezi různými SŘBD a nepružnost v rámci objektově-relačního problému. (Bauer a King, 2007)

Pro vyřešení těchto problémů je v rámci Hibernate frameworku představen objektově-orientovaný databázový jazyk pojmenovaný Hibernate Query Language (HQL). HQL je dotazovací jazyk pro vytváření dynamických a statických dotazů vyjádřených metadaty. HQL dotazy jsou zkompilované do standardních SQL dotazů podle SŘBD, ve kterých mají být využity. HQL je schválně navrženo, aby bylo svou syntaxí podobné SQL jazyku pro jednoduché pochopení a naučení, a tím se minimalizuje učící křivka. Pomocí HQL je tedy umožněno získávání persistentních objektů. HQL je taky možno chápat jako objektově orientovaná nadstavba jazyka SQL, jež snižuje rozdíly mezi objektově orientovanými systémy

a relačními databázemi. Ve druhé verzi Hibernate je možné využít HQL pouze pro získávání objektů, ale od třetí verze je možné již využívat manipulačních dotazů nad databázovými tabulkami. Tedy kromě získání lze i měnit, vkládat nebo mazat data. (Kumar, 2013)

2.1.6 Systém řízení báze dat Oracle

Systém řízení báze dat je úložiště dat ve formě tabulek. Relační systém řízení báze dat je podtypem ŠRBD, který používá data uložená ve formě tabulek, přičemž můžou být mezi nimi určeny vztahy. Navíc, je možno ukládat informace o tabulkách, jako počet a typy sloupců formou tabulek.

Obecný SŘBD lze specifikovat tím, že je programově spravován, většinou nepodporuje paralelní přístup více uživatelů databáze najednou, neexistuje podpora zabezpečení dat a distribuovaných databází, uživatelé mohou přistupovat přímo k uloženým souborům. V relačním typu ŠRBD jsou na druhou stranu relace mezi tabulkami uložené v databázi pouze formou tabulek, je to víceuživatelský systém, je v něm obsaženo více úrovní bezpečnosti, tabulky může být vytvořeny různými uživateli, uživatelé nemají přístup k databázovým souborům, existuje podpora distribuovaných databází, jsou podporovány abstraktní pohledy apod. V tabulce níže jsou uvedené jednotlivé produkty relačních SŘBD na trhu podle popularity vzestupně od nejpopulárnějšího po méně populární. Na prvním místě se v roce 2010 umístila Oracle Database od Oracle Corporation. (Asnani, 2010)

	Relační systémy báze dat
1.	Oracle Database od Oracle Corporation
2.	SQL Server od Microsoft
3.	DB2 od IBM
4.	Sybase od SAP
5.	Ingres
6.	Microsoft Access
7.	MySQL
8.	PostgreSQL

Tab. 2.1 Popularita jednotlivých SŘBD (Asnani, 2010)

Oracle Corporation se sídlem v Redwood Shores byla založena v 1977 a v roku 1979 se stává prvním dodavatelem, který nabízí komerční využití systému řízení báze dat. Software firmy Oracle je dostupný pro mnoho různých platforem, od osobních počítačů po velké sálové počítače pro objemné paralelní zpracování. Oracle zaručuje vysoký stupeň nezávislosti

na výrobci hardwaru. V tomto ŠRBD je obsaženo mnoho komponent. Základní komponentou je jádro. Tomu je svěřena udržování všech fyzických dat, jejich transport mezi pamětí a externím úložištěm, správa paralelního zpracování informací a poskytování izolace transakcí. Díky jádru je taky zajištěna reprezentace všech fyzických dat na logické úrovni jako relační tabulky. Důležitou částí jádra je optimalizátor, pomocí něhož je rozhodováno jak přistupovat k fyzickým datům časově efektivním způsobem, a které algoritmy mají být použity pro použití při tvorbě výsledků SQL dotazů.

Aplikační programy a uživatelé smí komunikovat s jádrem pomocí SQL jazyka. Jazyk Oracle SQL je téměř úplnou implementací standardu ANSI/ISO/IEC SQL:2011. Oracle hraje důležitou roli v procesu standardizace SQL jazyka už mnoho let. (Haan a kol., 2014)

Práce je zaměřená na Oracle Database verze 11g, která je částí relačního ŠRBD firmy Oracle. Ta se rozděluje na řadu edicí, díky kterým si zákazníci můžou zvolit produkt, jež bude vyhovovat přesně jejich požadavkům. Pro implementace ve velkém měřítku, obsahující všechny balíčky Oracle databázových funkcí a možností, poskytující možnosti pokročilého zabezpečení databáze díky podpoře Virtuální Privátní Databáze (angl. Virtual Private Database – VPD) a mnoho další funkcí je poskytnuto v Oracle Enterprise Edition. Oproti tomu použití Oracle Standard Edition je vhodné pro malé nebo středně velké podniky a implementace. Tato databáze může být nahrána na server nebo na klastr pomocí Real Application Clusters (RAC). Databáze Oracle Standard Edition One je nejvhodnější pro malé projekty. Podporuje jen dva výpočetní jádra a nepodporuje RAC. V neposlední řadě lze zmínit Oracle Personal Edition, kterou používají jednotliví vývojáři. Její použití vyžaduje vlastnění licence na rozdíl od Express Edition, ale jsou v ní poskytnuty všechny funkce stejně jako v Enterprise Edition. Poslední možností pro volbu je Oracle Express Edition, jež je dostupná bez poplatku pro uživatele operačního systému Windows nebo Linux. Je omezena 1 GB v paměti a 4 GB na disku. Je v ní poskytnuta podmnožina funkcí z Oracle Standard Edition jako zálohování a obnovení na serveru nebo automatickou správu úložiště. (Greenwald, Stackowiak a Stern, 2008)

2.1.7 JavaServer Faces

Mnoho společností poskytujících návrh a realizaci webových aplikací musí rozhodnout, v jakém jazyce budou aplikace tvořeny. Buďto jde o rychlý vývoj v Microsoft ASP.NET nebo intenzivní kódování pro podporu vysokého výkonu, který je poskytnut například díky Java Enterprise Edition. Druhá zmíněná varianta je velmi atraktivní řešení,

jelikož je vysoce škálovatelná, lehce přenositelná na další platformy a podporována mnoha výrobci. Na druhou stranu pomocí ASP.NET je umožněno vytvářet poutavé uživatelské rozhraní bez značného a složitého programování. Samozřejmě ideálním řešením by byla jejich kombinace, a přesně proto byla vytvořena knihovna JavaServer Faces (JSF), díky které je umožněn rychlý vývoj uživatelského rozhraní pro webové aplikace psané v Javě.

Pokud je vývojář seznámen s vývojem klientských aplikací, pak může považovat JSF jako knihovnu Swing pro aplikace běžící na serveru. V JSF je poskytnuto velké množství funkcí, které vývojáři JSP musí ručně vytvářet sami jako například navigace na stránce nebo validace. Proto technologie servletů a JSP jsou základem aplikace a knihovna JSF je už pouhým rozšířením. Tato knihovna má podobnou architekturu jako Struts framework, ale pro pochopení JSF samozřejmě není nutné znát jiné frameworky, pouze jazyk Java a značkovací jazyk HTML. V JavaServer Faces je obsažena:

- sada předem připravených komponent pro vývoj uživatelského rozhraní,
- programovací model řízený událostmi a
- komponentový model, který umožňuje vývojářům třetích stran přispívat dodatečné prvky.

Některé z komponent knihovny JavaServer Faces jsou jednoduché, jako třeba vstupní pole a tlačítka. Jiné jsou zase o něco komplikovanější, jako datové tabulky a stromy. V této knihovně je obsažen veškerý kód potřebný pro správu událostí a organizaci všech komponent. Programátoři aplikací tak můžou tyto záležitosti do jisté míry ignorovat a zaměřovat se více na vývoj aplikační logiky. (Geary a Horstmann, 2015)

2.1.8 JavaServer Pages Standard Tag Library

V knihovně s názvem JavaServer Pages Standard Tag Library (JSTL) je obsažena sada značek, které jsou používány k poskytnutí funkcionality, jako např. proces rozhodování, opakování, zacházení s XML formátem, internacionalizace a lokalizace, přístup k databázi, parsování a formátování čísel nebo dat apod. Pomocí JSTL je taktéž umožněno provádět operace se znakovými řetězci.

JSTL značky jsou rozdělené do čtyř různých skupin nazvaných core, xml, sql a fmt, stejně jako knihovna funkcí pro práci se znakovými řetězci (fn). Ve skupině značek core jsou obsaženy značky, které jsou používány pro větvení, opakování, zobrazování hodnot, nastavování a mazání proměnných z různých rozsahů, odchytávání výjimek apod. Taky jsou

zde obsaženy značky, které pracují s URL adresami. Ve skupině s názvem xml lze najít značky, díky kterým je poskytováno parsování, psaní a transformace XML. Skupina sql obsahuje značky, které lze použít pro práci s databází. V té je zahrnuto připojení k databázi, správa transakcí, stejně jako operace vybrání, vložení, změny a odstranění záznamů. Oproti tomu jsou ve skupině fmt obsaženy značky pro formátování a parsování dat a čísel, správa lokalizace a časových zón, kódování požadavku apod. Funkce ve skupině s názvem fn lze použít pro provádění určitých operací nad znakovými řetězci v JSP. Podrobněji lze skupinu core ještě rozdělit na další čtyři funkční oblasti. (Matha, 2013)

První oblastí jsou všeobecně použitelné značky, nejčastěji používané pro práci s proměnnými o různých rozsazích, např. zobrazení hodnoty proměnné na obrazovku - značka *out*, nastavení hodnoty příslušné proměnné, nebo vlastnost cílového objektu - značka *set*, odstranění proměnné - značka *remove* a odchycení výjimky - značka *catch*.

V druhé oblasti jsou obsaženy značky pro zpracování podmínek na JSP stránce. Konkrétně se jedná o zpracování obsahu značky, pokud parametr „test“ je pravdivý - značka *if*, poskytnutí podpory pro vzájemně se vylučující podmínky - značka *choose*, alternativy uvnitř předchozí značky - značka *when* a poskytnutí poslední alternativy výsledku podmínky - značka *otherwise*.

Oblast pro provádění iterací je třetí popisovanou oblastí. V JSTL jsou umožněny dvě důležité funkce pro provádění iterací. Jde o iteraci nad kolekcí objektů - značka *forEach* a iteraci nad řetězcovými výrazy oddělenými poskytnutými oddělovači - značka *forTokens*.

Poslední oblastí jsou značky pro nakládání s URL adresami. Jde o import obsahu na stránku ze zdroje poskytnutého URL adresou - značka *import*, přidání požadavkových parametrů k URL adrese - značka *param*, vytvoření URL adresy při použití vhodných přepisovacích pravidel - značka *url*, a přesměrování klienta na jinou stránku pomocí protokolu HTTP - značka *redirect*. (Layka, 2014)

2.1.9 PrimeFaces

Knihovna PrimeFaces je komponentová nástrojová sada nové generace umožňující snadnější vývoj webových aplikaci a konkrétně uživatelských rozhraní založených na technologii JavaServer Faces (JSF). Obecně lze říct, že jsou v ní obsaženy komponenty jako datové tabulky, grafy, tlačítka, odkazy, kalendáře, mřížky, stromy, vstupní komponent s uživatelsky přívětivými efekty (využívající knihovny jQuery) umožňující velmi jednoduché

používání technologie AJAX. Navíc jsou v ní poskytnuty prvky jako dokovací panel nástrojů, hlášení zpráv nebo komponenty pro mobilní webové aplikace.

Mezi vlastnosti této knihovny je možno řadit vysokou míru použitelnosti, sofistikovanost, pružnost a interaktivitu. Právě díky těmto vlastnostem je knihovna PrimeFaces významnou technologií pro mnoho JSF vývojářů, protože je pomocí ní poskytnuto mnoho výhod oproti standardním JSF komponentám.

V PrimeFaces je taktéž poskytnuta technologie nazvaná PrimePush, která umožňuje podobně jako technologie Ajax Push z knihovny ICEfaces asynchronní získávání serverových aktualizací na straně klienta v reálném čase. To znamená, že událost na serveru může být předána několika klientům, jako například stolním počítačům, mobilním zařízením nebo tabletům v tom stejném čase. (Hlavats, 2013)

2.1.10 Apache Maven

Definice Mavenu je rozdílná podle toho, s jakou perspektivou je vnímán. Většina uživatelů ho pojmenovává jako konstrukční nástroj, protože je díky němu umožněno zkompilovat programy pro nasazení ze zdrojového kódu. Stavební inženýři a projektoví manažeři můžou chápat Maven jako něco více komplexního, jako třeba nástroj projektového managementu. Nástroj s názvem Ant je zaměřen výhradně na zpracování, kompilaci, balení, testování a distribuci, ale díky Mavenu jsou poskytnuty ještě jiné funkce. Kromě dříve zmíněných funkcí je pomocí Mavenu umožněno navíc např. spouštět reporty, generovat webové stránky nebo usnadnit komunikaci mezi členy v pracovním týmu.

Apache Maven je tedy podle definice nástrojem projektového managementu, ve kterém je zahrnut objektový model projektu, sada standardů, životní cyklus projektu, systém správy závislostí a schopnost spouštět zásuvné moduly v předem definovaných fázích životního cyklu projektu. Ovšem pokud je vyžadováno pouze zkompilování programů kvůli nasazení, není třeba už využívat funkcí projektového managementu.

Jádro tohoto nástroje není příliš komplikované, jelikož umožňuje parsování několika XML souborů, vedení záznamů o životním cyklu a obsahuje pár pluginů. Byl vytvořen pro předání většiny odpovědnosti na sadu pluginů, které ovlivňují životní cyklus Mavenu. Většiny prací, typu kompilace zdrojového kódu, balení bajtkódu, nasazování na stránky a další činnosti, se odehrávají v zásuvných modulech. Samotný nástroj Maven, který je dostupný ke stažení, nedokáže vytvořit WAR soubor nebo spustit JUnit testování. Většina

postupů je implementována v zásuvných modulech, které jsou získány z úložiště (angl. Maven Repository). Je to tak zpracované hlavně proto, aby se minimalizovala velikost Mavenu nutná pro stažení. Tím, že tento nástroj získává závislosti a pluginy ze vzdáleného úložiště, je umožněno jeho univerzální opakované použití.

Projekt je definován skupinou identifikátorů, artefaktovým identifikátorem a verzí. Tyto informace jsou uloženy v objektovém modelu projektu (angl. Maven Project Object Model, zkr. POM). V tomto modelu jsou zapsány externí zásuvné moduly, jejich nastavení a přizpůsobení. Většina vývojových prostředí, jako Eclipse, NetBeans apod., mají tento model uložen v projektu na stejném místě. (Sonatype, 2014)

2.1.11 Apache Tomcat

Tomcat je bezplatný a plně funkční server (resp. kontejner pro servlety), pomocí kterého je umožněno vývojářům servletů a JSP spouštět a testovat jejich kód. Ale Tomcat není používán pouze jako server pro testování. Mnoho společností využívá tohoto serveru pro svá produkční prostředí, protože se prokázal být dost stabilní. Mezi firmy, které tento server využívají, lze řadit WalMart, General Motors, ale i běžné poskytovatele internetu, kteří poskytují hosting menším podnikům. Tomcat je používán pro reálné aplikace od malých fotoalb online (Webshots) po vysoce výkonné finanční webové aplikace (ETrade)[2]. Proto je jeho použití vhodné i v tomto projektu.

Navzdory popularitě Tomcat serveru v něm schází to, co schází většině open source projektů, a to nedostatek kompletní dokumentace. Určitá dokumentace je poskytnuta, ale přesto existuje vysoká potřeba pro další informace.

Mnoho tvůrců vytvořilo konkurenci Apache Tomcat svými aplikačními servery kompatibilními s aplikačním rozhraním Java EE. Kompatibilita v tomto významu znamená, že aplikační servery prošly mnoha testy pro zajištění úplné kompatibility. Tvůrci aplikačních serverů, kteří takovýmto procesem prošli, obdrželi licenci Java Enterprise Edition právě jako dokument potvrzující kompatibilitu.

Dva nejvíce používané komerční Java EE aplikační servery jsou WebSphere od společnosti IBM a WebLogic od společnosti BEA. Kromě těchto dvou existuje ještě mnoho dalších open source implementací, jako např.:

[2] Více společností využívající server Apache Tomcat pro své webové stránky nebo aplikace lze nalézt na adrese: http://wiki.apache.org/tomcat/PoweredBy

- JBoss,
- JOnAS,
- Geronimo a
- Glassfish.

V dnešní době je Apache Tomcat dostupný pro téměř všechny operační systémy, protože kromě zpřístupnění zdrojových kódů jsou dostupné i binární soubory pro více než desítku operačních systémů. Tomcat je klíčovou součástí většího celku Java EE standardů. Tyto standardy definují aplikační rozhraní založené na stejnojmenném programovacím jazyce, který je vhodný pro vytvoření webových aplikací. (Chopra, Li a Genender, 2007)

2.1.12 Subversion

Apache Subversion (obvykle zkracován pouze na „SVN“) je open source systém pro verzování zdrojového kódu a změnu konfiguračních příkazů jako nevázaného kódu, který je vytvořen pod licencí Apache. Vývojáři využívají Subversion pro zachování aktuálních a minulých úprav souborů jako programového kódu a obecnou podporu historie dokumentů. Cílem tohoto systému je býthlavně vhodnou náhradou v značné míře používaného Concurrent Versions System (CVS).

Obecně existuje mnoho distribuovaných systémů pro správu verzování nebo systémů správy zdrojového kód. Mezi ty, které jsou nejčastěji používány v posledních letech, lze řadit například:

- Concurrent Versions System (CVS),
- Apache Subversion (SVN),
- Git,
- Mercurial apod.

Subversion je poskytován široké veřejnosti díky projektům jako např. Apache Software Foundation, Free Pascal, FreeBSD, GNU Compiler Collection (GCC), Mono a SourceForge. Taktéž Google Code poskytuje služby Subversion pro jejich programy. V neposlední řadě i CodePlex nabízí přístup k SVN stejně jako k jiným službám. Tento systém byl vytvořen díky společnosti CollabNet v roce 2000 a dodnes je celosvětově používaným programem firmy Apache. (Darwin, 2014)

2.1.13 Eclipse IDE

Java programy lze psát v textových editorech a je možno je zkompilovat z okna příkazové řádky, ale toto není efektivní cesta vývoje softwaru. Profesionální programátoři používají vývojová prostředí (angl. Integrated Development Environment – IDE), ve kterém je zahrnut editor, kompilátor, kontextová nápověda, ladící program a další. Mezi populární Java IDE, lze řadit Eclipse, IntelliJ IDEA a NetBeans.

Eclipse je nejpoužívanějším vývojovým prostředím a vývojáři je používán pro kompilaci a spouštění Java programů. Přechod z jednoho IDE na druhé není těžkým úkolem, ale pokud programátor zjistí, že je díky danému IDE v určitých oblastech více efektnější než pomocí jiných vývojových prostředí, pak by měl zvolit to, ve kterém se mu pracuje nejlépe.

Vývojové prostředí Eclipse je open source produkt, který byl původně vytvořen společností IBM, s tím že ta předala značnou část kódu Java komunitě a od tohoto momentu se stal Eclipse tzv. komunitou řízeným produktem. Jeho původním cílem bylo poskytovat podporu programátorům vyvíjejícím v jazyce Java, ale v dnešní době je to vývojová platforma, která poskytuje tisíce nástrojů a zásuvných modulů.

Někteří vývojáři používají aplikační rozhraní tlustého klienta RPC pro vývoj uživatelských rozhraní. S vývojovým prostředím Eclipse není problémem vygenerovat a nasadit webové aplikace, spouštět a zastavit aplikační server, spravovat databázi jako administrátor a mnoho dalšího. Dále vývojáři používají jeho zásuvné moduly pro tvorbu reportů.

Kromě toho, že je Eclipse vývojovým prostředím, podporuje vývoj samotných zásuvných modulů, ale taky si vývojáři můžou vybrat pouze ty moduly, které zrovna potřebují nebo je chtějí vyzkoušet. V tomto IDE, se můžeme setkat se zásuvnými moduly typu zobrazování UML diagramů, systém hlášení zpráv nebo taky se zásuvnými moduly umožňujícími vývoj programů v jazycích jako C, JavaScript, Apache Flex a dalších. (Fain, 2015)

2.2 Návrhové vzory

Návrhové vzory poskytují řešení pro běžné problémy s návrhem aplikací. V objektově orientovaném programování jsou návrhové vzory používány spíše z důvodů tvorby a interakce objektů, než pro velké problémy způsobené softwarovou architekturou. Jsou díky nim poskytnuty řešení, která můžou být použity v každodenních aplikacích.

Podnikové vzory se liší od návrhových vzorů tím, že podnikovými vzory jsou řešeny stejnojmenné aplikace a jejich problémy, které se značně liší od desktopových aplikací. Service Oriented Architecture (SOA) je novým přístupem, díky kterému je představeno několik principů pro vývoj dobře organizovaného, znovu použitelného softwaru.

Návrhové vzory sice reprezentují souhrn znalostí, ale to neznamená, že je nutné je používat v každém čase. Významný americký psycholog řekl:

> *“Pokud je vaším jediným nástrojem kladivo,*
> *máte tendenci vidět každý problém jako hřebík.”*
>
> **- Abraham Maslow**

Pokud se vývojář snaží řešit všechny problémy pouze těmi návrhovými vzory, která zná, pak ne jenom, že nebudou fungovat, ale co je ještě horší, budou fungovat špatně, což častokrát způsobí ještě větší problémy. Navíc, při značném nadužívání návrhových vzorů začíná být systém až moc komplikovaný, což má za následek snížení výkonu. Jenom proto, že je u vývojáře oblíbený návrhový vzor „Dekorátor“ to neznamená, že ho musí používat na každý objekt. Vzory jsou využity efektivně tehdy, pokud je potřeba jejich užití. (Yener a Theedom, 2015)

2.2.1 Inversion of Control

Návrhový vzor Inversion of Control (IoC) lze nejlépe popsat hollywoodským pravidlem, které zní: „Nevolej nám, my zavoláme tobě.“ Tuto větu častokrát slyší mladí umělci od produkčních manažerů v Hollywoodu. Nicméně je tato věta důležitá i v rámci kontroly aplikačního toku při vývoji softwaru z důvodu zajištění vysoké soudržnosti a slabých vazeb. Pro lepší pochopení je dobré si představit případ, že metody v aplikaci provádí nějaké výpočty a výsledek je uložen pomocí logovací knihovny log4j. V případě volání metod z knihovny log4j je za kontrolu toku odpovědný aplikační kód.

Na druhou stranu je Inversion of Control základem pro každý framework. Při použití IoC je objekt dané aplikace zaregistrován frameworkem, který má na starosti volání metod zaregistrovaných objektů v příslušném čase nebo události. Tato kontrola je v takovém případě obrácená, protože namísto aplikačního kódu, který by měl spouštět metodu je to v tomto případě přesně naopak. Ve zkratce je tedy IoC pravidlem, které se týká povolení jinému objektu nebo frameworku volat metody aplikačních objektů při výskytu příslušných událostí.

Návrhový vzor Dependency Injection (DI) je pro některé vývojáře to samé jako Inversion of Control. Tento názor je nesprávný, jelikož jsou to dva rozdílné, ale přesto něčím podobné koncepty. Stejně jako je IoC zodpovědný za obrácení aplikačního toku, tak je i pomocí DI popsán způsob, jak jeden objekt nalézá jiné objekty, na kterých potřebuje zavolat potřebné metody. Existuje několik způsobů jak dosáhnout Dependency Injection a jednou ze strategií k dosažení tohoto cíle je právě Inversion of Control. Dalšími z možností můžou být přímé vytváření instancí a využití strategie pomocné továrny, která je založena na návrhovém vzoru Tovární metody (angl. Factory Method). Ta zvyšuje vhodnost použití nového operátoru a v závislosti na vstupu poskytuje odpovídající instance objektů.

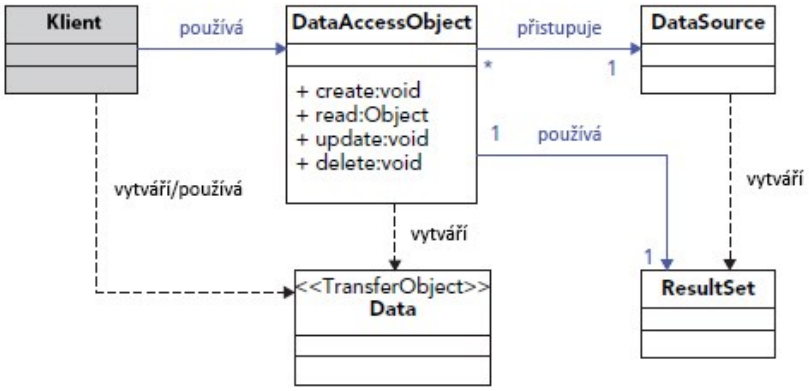
Mezi výhody Dependency Injection lze zařadit prosazování volných vazeb mezi objekty, jelikož už není třeba psát závislosti směrem od programu k rozhraní, ale je možno je konfigurovat mimo aplikaci a lze tak tvořit lehce zaměnitelné a propojitelné implementace. Přináší výhodu i při testování objektů, protože v tomto případě není vyžadován žádný spuštěný kontejner. Objekty můžou být testované tak dlouho, dokud jsou jim nějakým mechanismem vkládány závislosti.

Jako nevýhoda DI může být považován fakt, že závislosti jsou uchovávány v konfiguračních XML souborech, které jsou proprietární. Navíc propojování vysokého počtu různých instancí může být dost riskantní, pokud je třeba řešit velký počet závislostí. Závislosti založené na XML metadatech nebo nadměrné používání reflexe může vést ke snížení výkonu aplikace. (Kayal, 2008)

2.2.2 Data Access Object

Obecným cílem návrhového vzoru Data Access Object (DAO) je zapouzdřit způsob přístupu k datům tím, že je pomocí něho poskytnuto rozhraní, díky kterému můžou komunikovat různé vrstvy se zdrojem dat. Je pomocí něho poskytnutá správa propojení s datovým zdrojem pro získávání a ukládání dat.

Problém, který byl vyřešen abstrakcí a zapouzdřením datového zdroje, se týkal závislosti aplikace na implementaci datového zdroje. Toto řešení poskytlo oddělenost logické a databázové vrstvy a mělo negovat jakýkoliv dopad problému, který vzniká při změně trvalého úložiště. Nicméně ve skutečnosti se databázové systémy v historii moc neměnily, dokonce ani mezi výrobci stejného typu jako třeba PostgreSQL a MS SQL. Je těžké si představit, že by obecně ve společnostech byly schvalovány přechody ze zdroje SQL na XML ploché soubory, LDAP repositáře nebo webové služby.



Obr. 2.4 Diagram tříd zobrazující návrhový vzor DAO (Yener a Theedom, 2015 – volně přeloženo)

Data Access Object je stále hodnotným návrhovým vzorem a jeho původní řešení je stále platné, i když se změnil jeho účel. Namísto řešení důsledku nepravděpodobné změny datového zdroje, jeho výhoda činí hlavně ve strukturování kódu, čímž je udržován přehledný kód přístupu k datům. Stále existují ještě výhody v zapouzdření systému pro ukládání dat nebo zjednodušení přístupu ke komplexním implementacím datových zdrojů. Nicméně tyto přínosy jsou pro vývojáře spíše výjimky.

Návrhový vzor DAO zapouzdřuje základní operace s datovým zdrojem v rámci rozhraní, které je implementováno konkrétní třídou. Toto rozhraní je lehce testovatelné, bez nutnosti připojení k databázi. Konkrétní Data Access Object implementace používají aplikační rozhraní jako třeba Java Persistent API a Hibernate pro vykonávání základních operací nad úložištěm dat. (Yener a Theedom, 2015)

2.2.3 Model-View-Controller

Softwaroví inženýři musí často ve svých projektech implementovat podobné architektonické řešení. V průběhu let bylo publikováno mnoho návrhových vzorů ve formě ať už elektrických, či tištěných knih. Jedním z těchto návrhových vzorů je Model-View-Controller (MVC). Hlavní myšlenkou tohoto vzoru je oddělit kód pro uživatelské rozhraní (pohled), ukládání dat (model) a zpracování komunikace, či dalších činností mezi pohledem a modelem (řadič). Implementace MVC se liší podle druhu aplikace, ale základ zůstává ten samý.

Dokonce použití tak běžné komponenty jako tlačítko může pramenit v dost složitou interakci mezi jednotlivými vrstvami. Samozřejmě po vizuální stránce nemusí činnost vypadat vůbec složitě, ale pro programátora už to tak snadný úkol není. Ten se musí postarat v případě MVC architektury o kódování uživatelského rozraní, modelu a potřebné funkcionality.

V případě tohoto vzoru se musí v praxi programátoři vyvarovat přílišného kódování funkcionality v uživatelském rozhraní a modelu. Kód pro poskytnutí dat z modelu a naopak poskytnutí dat z pohledu by měl zůstávat výhradně v rámci řadičů. (Fain, 2015)

2.3 Metodiky vývoje softwaru

Správy vývoje softwaru je dosaženo hierarchickou dekompozicí prací z vysoko úrovňových na nízko úrovňové činnosti. Aktivity na nejnižší úrovni jsou pak pojmenovány úlohy, a z toho se dá odvodit, že činnosti jsou agregátem úloh. Pro systematické provádění úkolů je potřeba následovat sadu postupů a používat různé nástroje a techniky.

Procesní inženýrství se zaměřuje na vývoj softwaru a neustále je díky němu zlepšován pracovní tok mezi vývojovými úlohami, aby byly více účinné a efektivní, tj. splnit úlohu bez ztráty času, snahy a zdrojů, přičemž je dosaženo požadovaných výsledků. Zdokonalené pracovní procesy vedou ke zvýšení produktivity a morálky softwarových vývojářů, jakosti produktů a spokojenosti uživatelů a zákazníků. Výsledkem byly vytvořeny a jsou i nadále tvořeny a upravovány metodiky vývoje softwaru.

Vývojář nemusí striktně následovat metodiky vývoje, protože slouží hlavně jako jakési pomocné vodítko, které ukazuje směr a způsob, ale záleží na vývojáři, jakým způsobem je metodika využita nebo je upravena pro potřeby konkrétního projektu. Tyto metodiky rozdělujeme na tradiční a iterativní. Jako nejznámější tradiční metodiku vývoje softwaru lze považovat vodopádový přístup, který byl vytvořen a představen Winstonem Roycem v roce 1970. K iterativním modelům je možno počítat přírůstkový, evoluční, agilní a spirálový přístup. (Fairley, 2009)

Rational Unified Process (RUP) od společnosti IBM je metodika pro iterativně-přírůstkový vývoj softwaru. Díky ní je poskytnut přístup k přidělování úloh a povinností uvnitř organizace zabývající se programováním aplikací. Cílem je zajistit vývoj vysoce kvalitního softwaru, který plní požadavky uživatelů při dodržení časového plánu a rozpočtu.

Pomocí RUP je popsáno efektivní použití komerčně osvědčených přístupů v rámci vývoje softwaru. Jejich výhoda není ani tak v tom, že je možno kvantifikovat jejich hodnotu, ale spíše v tom, že jsou často používány v praxi úspěšnými organizacemi. V této metodice jsou poskytnuty pokyny, modely a nástroje pro získání výhod jako iterativní vývoj, správa požadavků, použití komponentové architektury, vizuální zobrazování modelů, ověřování softwarové kvality a řízení změn softwaru. Jeden z nástrojů je i případ užití, který zobrazuje

vztahy účastníků s modelovaným systémem. Pomocí tohoto nástroje si lze jednoduše představit hlavní funkce systému a jednotlivé pravomoci. Je rovněž možno využívat i ostatních nástrojů z jazyka Unified Modeling Language (UML) pro modelování systému. (Barnes, 2007)

Agilní metodiky jsou moderní přístupy k tvorbě nového softwaru založeného na spolupráci se zákazníkem, týmové práci, iteračním vývoji a reakci na změnu. Díky kombinaci mezilidské komunikace s flexibilními metodikami je dosaženo optimálního výsledku, který je modelován formou uživatelských příběhů a rychlým vývojem funkčního softwaru. Nicméně klíčem k agilním metodikám jsou časté komunikace se zákazníkem spolu se soudržnou týmovou spoluprací. Mezi agilní metodiky je možno řadit Scrum, Extrémní programování, dynamický vývoj systémů (angl. Dynamic Systems Development) a mnoho dalších. (Rico, Sayani a Sone, 2009)

Extrémní programování (XP) je takovou metodikou vývoje softwaru, ve které jsou všechny praktiky zahnané do extrémů. Mezi klíčové vlastnosti lze řadit časové uspořádání, párové programování, programování řízené testy, programování komponent až tehdy, když jsou požadovány, plochou řídící strukturu, jednoduchost a jasnost kódu, očekávání změn v požadavcích a častou komunikaci se zákazníkem.

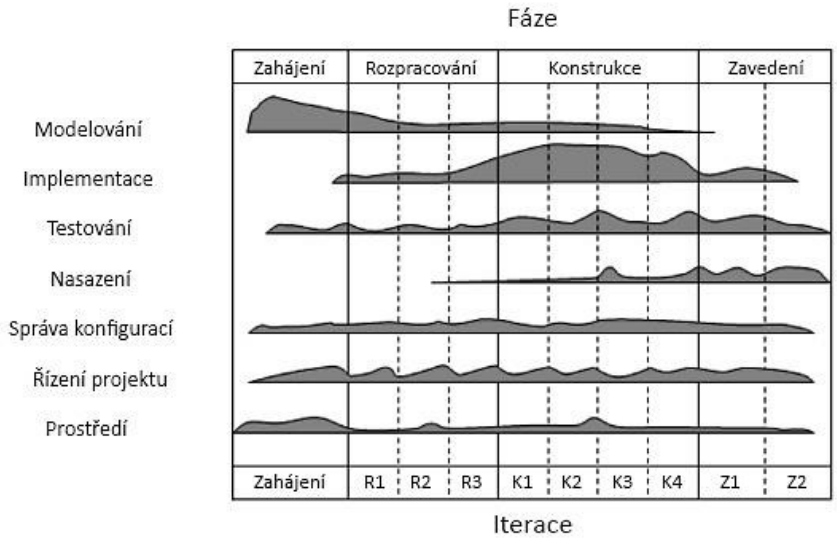
Pojem extrémní je v této metodice použit z důvodu netolerování aktivit, které nemají okamžitý přínos. Takže například přidání sloupce do tabulky z důvodu možného využití v budoucnu není vykonán, protože v daném momentu zvyšuje náklady a pravděpodobnost možných komplikací bez žádného okamžitého přínosu. Existuje taky významné riziko, že přidaný sloupec nebude nikdy v budoucnu využit. Extrémní strategií je v tomto případě přidání sloupce, až bude vyžadován. (Cimolini a Cannell, 2012)

Agile Unified Process (AUP) je agilní verzí obecného Unified Process (UP), se vznikem v září roku 2005. Autorem této metodiky je Scott Ambler, který mj. napsal několik knih na téma agilního vývoje a UP. Tato metodika následuje všechny čtyři fáze Unified Process a to zahájení, rozpracování, konstrukce a zavedení. Rozdíl je v tom, že každá z těchto fází je prováděna iterativně.

Pracovní postup je rozdělen do sedmi částí (pravidel) a to **modelování**, ve kterém je snaha o pochopení nejen problému a jeho omezení, ale i předpokladů a možných řešení, jejímž výsledkem by měl být model. **Implementace**, při níž je model přeměněn na spustitelný program, přičemž je rovněž provedeno základní testování, například testování jednotek

(angl. Unit testing). **Testování** je částí, ve které je zahrnuto podrobnější testování a hledání chyb, ověření implementace a uživatelské testování. **Nasazení,** je část, ve které se odehrává proces doručení výsledku, tedy příprava pracovního prostředí, trénování a instalace. Další částí je **správa konfigurací**, v níž je obsažena správa položek vyprodukovaných systémem tedy dokumentace, návrhy a zdrojový kód. Jako předposlední část lze brát **řízení projektu**, během kterého se řídí celý projekt a to zmírňování vzniklých problémů, přiřazování úloh a pozorování chodu. Poslední je **prostředí**, kde je kladen důraz na poskytnutí potřebného hardwaru, softwaru, tréninku, manuálů a administrativní podpory.

Častokrát běží všechny tyto části simultánně, ale s různým rozsahem působení. Například na začátku projektu hraje fáze modelování velmi důležitou roli, přičemž část nasazení prakticky neexistuje. Grafické znázornění je možno vidět na obrázku níže, přičemž si lze všimnou i dílčího rozdělení některých fází.



Obr. 2.5 Intenzita jednotlivých částí během životního cyklu AUP (Stephens, 2015 - volně přeloženo)

Hlavním rozdílem mezi Agile Unified Process a obecným UP je v tom, že obecný Unified Process má za následek vydání pouze jedné verze. Následováním metodiky AUP je vyprodukováno mnoho verzí, díky agilnímu iteračnímu přístupu. Většina iterací není nasazena hned do produkčního prostředí, ale je možno tyto verze používat jako demonstrační program pro koncového uživatele. (Stephens, 2015)

Mezi filozofie metodiky AUP patří informovanost zákazníka, jednoduchost, agilnost, zaměření na důležité aktivity, nezávislost nástrojů a úprava metodiky pro vlastní potřeby. Právě nezávislost nástrojů je důležitou vlastností, protože díky ní lze kombinovat Use Case 2.0 (RUP) např. s uživatelskými scénáři (XP). (Cimolini a Cannell, 2012)

3 Analýza a návrh systému v neziskové organizaci

3.1 Analýza současného systému organizace

Informační systém včetně webové aplikace je tvořen pro vyhovění požadavku neziskové organizace „Slezská diakonie“ poskytující hlavně služby v sociální oblasti. Organizace působí hlavně v Moravskoslezském kraji, ale v poslední době rozšířila svou činnost i do Olomouckého a Jihomoravského kraje. Celkově čítá sto čtyři zaregistrovaných sociálních služeb jako např. azylové domy, domovy pro seniory, nízkoprahové denní centra, pečovatelské služby, sociálně terapeutické dílny apod.

Slezská diakonie má jako organizace sepsanou rámcovou smlouvu s akciovou společností T-Mobile, a.s. pro podnikové sítě, která ji umožňuje využívat množstevních, speciálních slev na hlasové služby, mezinárodní volání a roaming, datové služby a internet. Tyto speciální tarify a slevy jsou poskytovány zaměstnancům Slezské diakonie a jejich známým (max. 5 uživatelů na jednoho zaměstnance). Mezi výhody lze započítat volání zdarma mezi uživateli této podnikové sítě a taky zvýhodněné ceny jiných volání.

Současný systém správy mobilních smluv je čistě v papírové formě, vedený v sešitech. V tomto systému jsou nejen vedeny smlouvy, ale taky seznamy dlužníků, kteří platí po splatnosti nebo mají dluhy měsíc po splatnosti. Hlavní problém je v přehlednosti, jelikož uživatelů této sítě je už více než šest tisíc. Dalším problémem je nečitelnost některých údajů, protože smlouvy a další dokumenty jsou připraveny formou šablon, ale po vytisknutí jsou stejně vyplňovány ručně psaným písmem, a proto nejsou vždy čitelné. Rovněž se stává, že papírové smlouvy se ztrácí, nejsou dobře evidovány, uživatelé privátní podnikové sítě si stěžují, že jim smlouva poštou nepřišla, i když byla odeslána a doručena už pár týdnů apod.

Požádat o zařazení do privátní podnikové sítě (PPS) můžou zaměstnanci organizace. Kromě pracovníků můžou podat žádost o přijetí do sítě i běžní občané ČR, ovšem pod podmínkou, že za ně bude nějaký zaměstnanec ručit. Počet osob v registrovaných u jednoho zaměstnance je omezený na pět. Do této sítě se může připojit zaměstnanec, který už není ve zkušební lhůtě a není s ním vedeno řízení o ukončení pracovního poměru.

Existují tři způsoby, jakým může zaměstnanec, resp. člověk registrovaný pod ním, vstoupit do sítě PPS. Prvním způsobem je vyplnění „Dohody o datu přenesení čísla“. Ten se uplatňuje v těch případech, kdy žadatel má číslo u jiného operátora než T-Mobile. Pro každou smlouvu je vygenerované unikátní čtrnáctimístné Číslo Výpovědi Opouštěného

Poskytovatele (ČVOP). Toto číslo musí žádající osoba nahlásit svému současnému operátorovi pro uvolnění čísla. Po zaslání vyplněného formuláře je společností T-Mobile následně zasláno datum přenosu spolu s účastnickou smlouvou. Tento způsob trvá v porovnání s ostatními obvykle nejdéle. Převod je uskutečněn do jednoho měsíce od potvrzení o přijetí formuláře.

Druhý způsob se nabízí pro žadatele, který už má vytvořenou a sepsanou smlouvu u T-Mobile a tak dojde pouze ke změně zákaznické smlouvy na rámcovou. V tomto případě je vyžadované vyplnit „Dohodu o převodu účastnických smluv“ včetně základních informací, rodného čísla, čísla zákaznické smlouvy a podpisu. Uchazeč o smlouvu musí smlouvu podepsat a zaslat/zanést zpět na adresu organizace do čtrnácti dnů. Při využití tohoto způsobu připojení do sítě PPS je převod uskutečněn do tří dnů od potvrzení o přijetí formuláře a zaslání čísla účtu kontaktním pracovníkem organizace.

Pokud žadatel nemá žádné číslo, nebo má předplacenou kartu (Twist) musí vyplnit dokument „Účastnická smlouva LE hromadný“. Při převodu z Twistu je nutno dodat rovněž 19místné výrobní číslo karty „Integrated Circuit Identity“ (ICCID), které je uvedeno na spodní straně SIM karty. Tento způsob trvá taktéž tři pracovní dny a tedy stejně jako způsob, kde dochází k převodu zákaznické smlouvy na rámcovou.

3.2 Analýza požadavků

Požadavky získané po konzultaci s kontaktní osobou v organizaci jsou rozdělené na funkční a nefunkční požadavky, na základě kterých je založen návrh systému.

Mezi funkční požadavky lze řadit:

- vytvoření eletronické evidence uživatelů PPS,
- automatické hlídání a upozornění na potřebné prodloužení smlouvy,
- automatické vyplňování smluv,
- automatické generování PDF souborů ze smluv,
- hlídání dlužníků a automatické upozorňování e-mailem,
- automaticky vyplněná hromadná korespondence,
- získání vygenerované smlouvy v aplikaci a přes e-mail,
- zrušení smlouvy,
- autorizace zaměstnanců/uživatelů žádajících o novou smlouvou a následné zaslání potvrzovacího e-mailu,
- možnost vygenerování hesel všem uživatelům při zavedení aplikace apod.

Dílčím požavavkem aplikace je psaní novinek týkajících se ať už aplikace nebo změn smluv administrátorem na úvodní obrazovky uživatelů. Dalším je nahrání naskenovaných smluv do systému, jejích úplnou správu a kontrolu nad zasláním a přijetím smlouvy. Funkčních požadavků není konečný počet, protože i po dokončení určité verze aplikace můžou vzniknout nové požadavky na aplikaci. Proto byla rovněž zvolená agilní metodika vývoje – Agile Unified Process.

Nefunkčními požadavky jsou v případě řešeného systému výkonnostní efektivita, spolehlivost, přenositelnost a bezpečnost. Přenositelnost zajišťuji multiplatformní jazyk Java, který umožňuje aplikaci nasadit na téměř jakémkoli typu aplikačního serveru. Požadavky výkonností efektivity a bezpečnosti jsou v rámci aplikace testovány a optimalizovány podle potřeby ke konci práce.

3.3 Návrh nového systému

Ve fázi návrhu je výběr technologií pro následující fázi realizace důležitou úlohou. Je třeba zvolit takovou kombinaci jednotlivých verzí, které jsou mezi sebou kompatibilní a splní svůj účel při programování aplikace. Tato volba není jednoduchá, protože při pokusu o použití všech nejnovějších verzí daných technologií si lze velmi lehce všimnout, že některé technologie byly naposledy aktualizovány před dvěma lety a jiné zase před pár týdny. Tento časový nesoulad vede k nekompatibilitě a mnoha chybám.

Pro tuto práci byly zvolené technologie ve verzích Java Enterprise Edition 7, JDK 8, Spring Framework 3.2, Oracle Database 11g Express Edition, Hibernate 4.3, JavaServer Faces 2.1, PrimeFaces 3.4, Apache Tomcat 8, Log4j 1.6 a jUnit 4.8. Při pokusu o aktualizování verze jakékoli technologie byla vždy vyhozena chybová hláška z důvodu nekompatibility jednotlivých knihoven. Mít přehled o všech změnách a verzích je dost náročný úkol, který vyžaduje časté kontroly diskuzních fór a oficiálních stránek daných knihoven, a proto je pro programátora častokrát lepší, pokud je právě tento úkol přidělen např. softwarovému architektu nebo obdobné pozici, která má na starosti analýzu a návrh systému.

3.3.1 Vstupy

Ke vstupům systému lze řadit seznam zákazníků privátní podnikové sítě neziskové organizace, přičemž z toho většinu tvoří zaměstnanci, ale ti mají možnost připojit do sítě až pět soukromých mobilů. Při připojování nového čísla musí zaměstnanec zvolit odpovídající

smlouvu a to buď „Účastnickou smlouvu LE“, „Dohodu o datu přenesení telefonního čísla“ nebo „Dohoda o převodu účastnických smluv“. Účastníci systému se rozdělují na tři role: uživatel, asistent a administrátor. Administrátor smí psát novinky aplikace viditelné všem zaměstnancům. Tyto popisy velmi usnadňují návrh databáze v pozdější fázi projektu. Počáteční vstupy lze rozdělit do skupin:

Uživatel: identifikační číslo uživatele, jméno, příjmení, e-mail, heslo, město pracovního střediska, datum narození, adresa, číslo bankovního účtu a název banky,

Role: identifikační číslo role a název role,

Novinky: identifikační číslo novinky, nadpis, obsah a datum,

Smlouva: identifikační číslo smlouvy, staré mobilní číslo, datum zavedení žádosti, datum přijetí žádosti, tarif, fakturační skupina, podrobnosti fakturační skupiny, typ vyúčtování služeb, způsob úhrady, roamingový tarif, hlasové a datové roamingové zvýhodnění, datový roaming limit, tarifní zvýhodnění, zpřístupnění gprs, edge a 3g, datové tarifní zvýhodnění, podrobný výpis služeb, blokace mezinárodních hovorů, typ blackberry, povolení blackberry roamingu, multimediálních zpráv, audiotex a prémiových sms, dms a sms plateb, m-plateb, stahování tónů a datum aktivace sim karty,

- **Přenesení tel. čísla**: číslo výpovědi opuštěného poskytovatele (ČVOP) a kód přenosu čísla (KPČ nebo PAC),
- **Převod účastnických smluv**: číslo zákaznické smlouvy, rodné číslo, potvrzení o převodu sms zprávou, přístupy do systému společnosti T-Mobile a poznámky,
- **Účastnická smlouva LE**: typ objednávky, typ sim karty, typ zařízení, minimální měsíční plnění, cena zařízení a výrobní číslo karty vkládané do poznámky.

3.3.2 Výstupy

Mezi výstupy systému týkající se zaměstnanců lze počítat přehled zaregistrovaných osob, za které ručí, přidání, úprava a zrušení takovéto osoby, při přidání výběr ze tří možností pro připojení čísla k síti, vygenerování smlouvy ve formátu PDF nebo DOC a zaslání na e-mail k podpisu, možnost nahrání podepsané smlouvy zpět do systému, pozorování procesu přijetí a schvalování smlouvy, případně se podívat na novinky aplikace. Asistenti administrátorů můžou tvořit uživatelské účty zaměstnanců včetně nahrání naskenovaných smluv. V případě výstupů systému pro osoby s administrátorskými právy jsou to stejné funkce jako u asistentů, ale k tomu přehled a správa všech zaměstnanců, příp. osob připojených k síti

právě díky nim, vygenerování hesel jednotlivě a hromadně, přijímaní podepsaných smluv a správu nad přijatými smlouvami včetně času přijetí a času vygenerování.

3.3.3 Funkce informačního systému

Funkce systému vychází z analýzy požadavků a definice výstupu navrhovaného systému. K přehlednějšímu zobrazení jednotlivých uživatelských rolí, funkcí a vztahů mezi nimi je vytvořen diagram případů užití (viz Příloha č. 1 – Diagram případů užití). Ten znázorňuje chování systému z hlediska různých účastníků. (Fowler, 2009)

V tomto diagramu lze vidět tři účastníky, u kterých se přepokládá alespoň do jisté míry určitá komunikace se systémem. Mezi tyto aktéry je možno považovat administrátora, asistenta a běžného uživatele (v tomto případě zaměstnance organizace). Asistent nemusí být nezbytně zaměstnanec. Předpokládá se, že asistenty můžou být studenti, kteří přijdou vykonávat odbornou praxi v řešené organizaci. Tuto roli může rovněž zastávat asistent vedoucího oddělení informatiky. U administrátora se už ale předpokládá, že je vedoucí oddělení informatiky.

Jednotlivé případy užití jsou popsány podrobněji pomocí jejich specifikace. Ta obsahuje identifikační číslo a název případu užití, účastníky hrající nějakou roli, vstupní podmínky, tok událostí a následné podmínky. V této práci jsou více popsané a specifikované případy užití: Registrace do systému (UC_02), Připojení soukromého mobilu (UC_06), Zaslání e-mailové zprávy (UC_10) a Hromadné vygenerování hesel (UC_16). Tyto specifikace byly vybrány nejen z důvodu značné různorodosti v rámci toku událostí a rolí, ale i částečné provázanosti a rozlišných vstupních, či výstupních podmínek (viz Příloha č. 2 – Vybrané specifikace diagramu případů užití).

3.3.4 Uživatelské scénáře

Většinou je naprosto zbytečné kombinovat případy užití a uživatelské scénáře tvořené samostatně, protože jsou vzájemnými substituty, používanými v různých vývojových metodikách. Jenže díky nové specifikaci případů užití druhé verze, je to právě naopak. Uživatelské scénáře tvořené na základě malých částí případů užití tvoří velmi dobré komplementy. Diagram případů užití a jeho specifikace tvoří kvalitní dokumentaci, která se může hodit v pozdějších fázích projektu a to je mnohem lepší, než velký počet víceméně nesourodých uživatelských scénářů. Díky případům užití je položen základ uživatelským

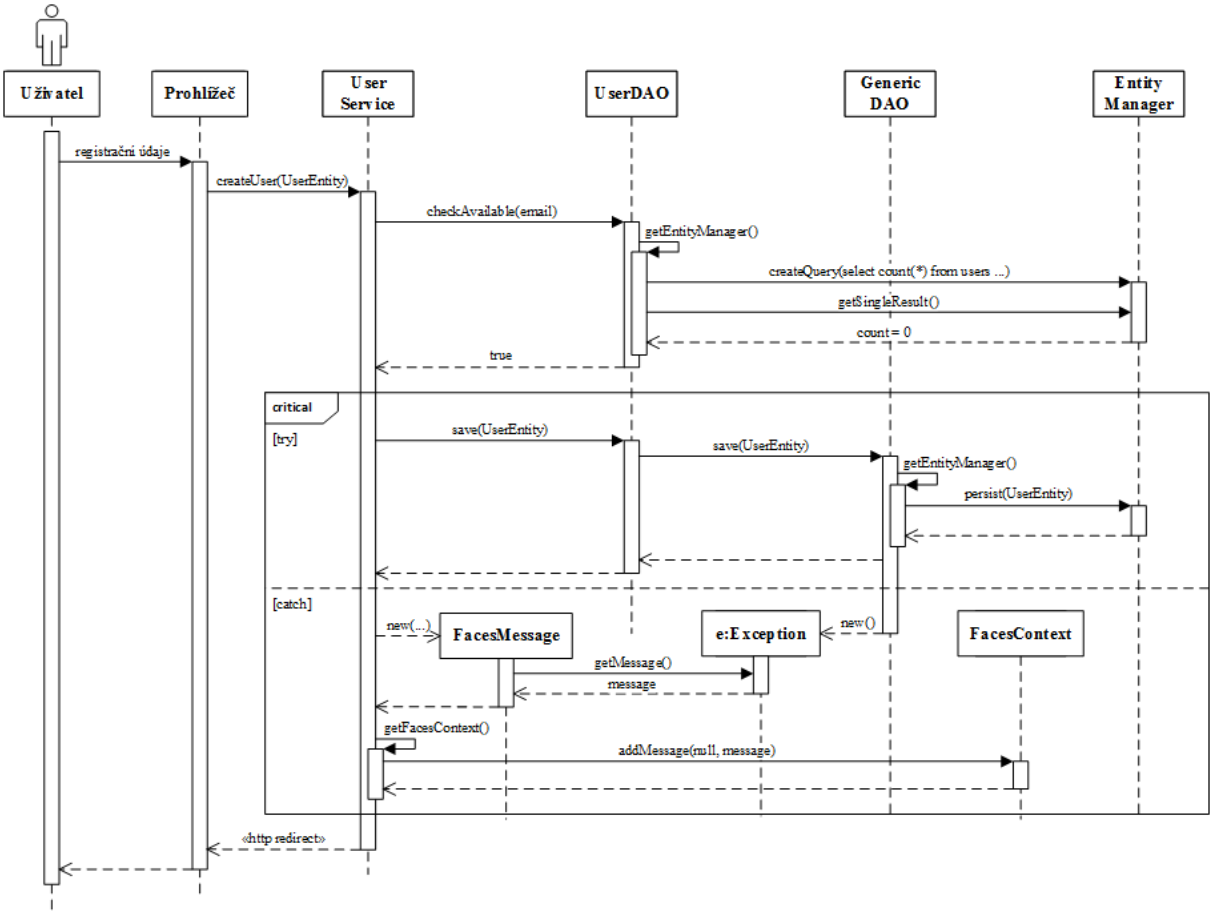
scénářům, které dobře reflektují modelování reality a poskytují jednodušší pochopení často složitých případů užití. Uživatelské scénáře v tomto projektu jsou:

- „Jako zaměstnanec chci připojit mobilní čísla k privátní podnikové síti, protože budu efektivněji a levněji komunikovat nejen s kolegy, ale i rodinou a známými“,
- „Jako zaměstnanec chci mít přehled o připojených číslech k síti, jelikož mi bude poskytovat usnadnění nad jejich správou a kontrolou“,
- „Jako zaměstnanec chci mít možnost zaslat nové heslo na e-mail, abych nemusel kontaktovat administrátora aplikace v případě, že bych stávající zapomněl“,
- „Jako asistent administrátora chci tvořit zaměstnanecké účty, neboť tím urychlím zavedení aplikace do organizace“,
- „Jako administrátor chci psát novinky aplikace, protože mi můžou poskytnout snadnější sdělování potřebných informací jejím uživatelům“
- „Jako administrátor chci mít přehled o všech zaměstnancích a připojených mobilech do sítě, abych v případě potíží mohl poskytnout podporu nebo potřebné důkazy“,
- „Jako administrátor chci schvalovat uživatele a žádosti o připojení čísla k PPS, z důvodu ochrany systému před neoprávněným užitím“ a
- „Jako administrátor aplikace chci mít možnost vygenerovat všem uživatelům aplikace nová hesla, aby byla urychlená fáze zavedení aplikace“.

3.3.5 Procesy a vazby navrhované aplikace

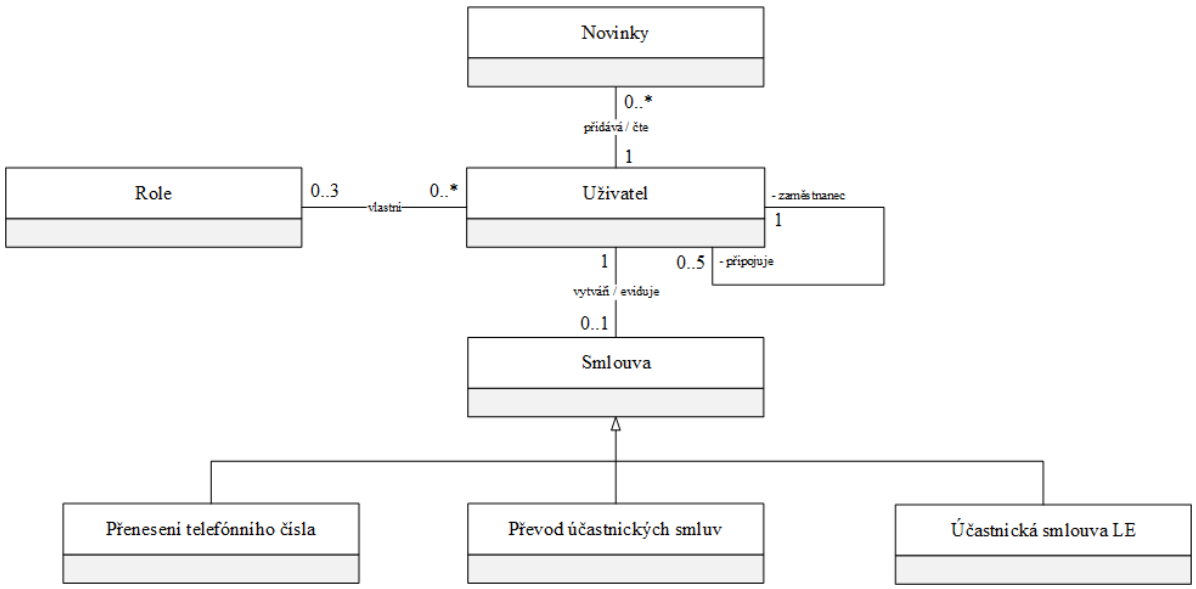
Pro zobrazení procesů v rámci modelovaného systému je vhodné využít sekvenčního diagramu. Pomocí tohoto diagramu je možno sledovat komunikaci mezi jednotlivými třídami a zasílání zpráv v případě, že je jejich přítomnost vyžadována. Samotný diagram slouží jako vhodný přehled pro pochopení návaznosti komponent mezi sebou navzájem. (Fowler, 2009)

Pomocí níže zobrazeného sekvenčního diagramu je znázorněn simplifikovaný proces registrace nového uživatele bez řešení transakcí a hashovaní hesla kvůli přehlednosti. Je nutno podotknout, že zaměstnanec se po takovéto registraci stále nemůže přihlásit do aplikace, protože čeká, až ho administrátor ověří pomocí jeho registračních údajů a schválí jako platného uživatele. Při schválení jsou přiděleny uživateli i uživatelská role, případně role, pokud by šlo například o asistenta nebo dalšího administrátora.



Obr. 3.1 Sekvenční diagram - Registrace nového uživatele

Díky diagramu tříd je umožněno vidět jednotlivé třídy, jejich strukturu a vztahy. Pomocí diagramu níže jsou zobrazeny hlavní business třídy systému sloužící k jednodušší realizaci aplikace a konkrétně tříd.



Obr. 3.2 Diagram tříd - Struktura systému

3.3.6 Návrh databáze

Dobrým způsobem jak znázornit návrh databáze v tomto projektu je e-r diagram (viz Příloha č. 3 – Entity Relationship Diagram). Pomocí něho jsou zobrazeny navrhované entity a vztahy mezi nimi. Pro namodelování tohoto diagramu byl použit program SQL Developer Data Modeler od firmy Oracle.

V návrhu databáze je využito různých typů kardinalit a to konkrétně nejen vztahy jedna k více a více k více, ale i unární vztah a generalizace. Dříve byla generalizace řešena buďto spojením tabulek do jedné nebo případně zabezpečení triggery před vložením záznamu pro zajištění integrity. V tomto projektu není využita ani jedna z těchto možností, protože projekt využívá možností objektově relačního mapování frameworku Hibernate.

V případě řešeného projektu je díky kombinace technologií JPA a Hibernate umožněno mapování dědičnosti tříd na relační tabulky. Existují tři možné přístupy této problematiky a to:

- jedna tabulka pro všechny třídy hierarchie,
- jedna tabulka pro jednu podtřídu a
- jedna tabulka pro jednu třídu.

V případě, že by byla zvolena první varianta, musí být v sloupcích dané tabulky povolené nulové hodnoty, jelikož ta bude obsahovat atributy všech entit spadajících do hierarchie dědičnosti. V druhém případě jsou tvořeny tabulky pouze pro podtřídy a atributy nadtřídy se opakují ve všech podtřídách. Pomocí posledního případu je ukázáno řešení, kdy jedna třída je mapovaná na jednu tabulku, ať už jde o nadtřídu nebo podtřídu v řešené hierarchii. A právě poslední řešení je zvoleno jako nejvhodnější pro tento řešený projekt.

Nutností při návrhu databáze není jen e-r diagram, ale taktéž datový slovník (viz Příloha č. 4 – Datový slovník). Jelikož je v této práci e-r diagram dosti podrobně vypsaný, včetně primárních a cizích klíčů, příp. datových typů, je otázkou zda má datový slovník vůbec smysl. Odpovědí je má, protože kromě těchto vlastností obsahuje ještě popis dané položky entity, což je v tomto projektu český překlad anglického atributu entity. Anglické atributy jsou zvoleny z důvodu možné budoucí spolupráce na tomto systému s osobami ze zahraničí, pro které je tak pochopení databáze o to snadnější.

4 Realizace, testování a zhodnocení systému

Tato část diplomové práce podrobně popisuje vývoj aplikace, její následné testování a na závěr kapitoly zhodnocení vytvořeného systému. V první části je kladen důraz na praktický popis tvorby, včetně všech nejdůležitějších součástí a propojení teoretických znalostí s jejich praktickým uplatněním.

4.1 Tvorba systému

Realizace aplikace podle metodiky Agile Unified Process vyžaduje iterativní programování založené na předchozí analýze a návrhu, použití verzování aplikace a časté konzultace vytvořené verze se zákazníkem. Tyto kroky nejsou v textu psány iterativně z důvodu přehlednosti, ale sekvenčně, i když je práce realizována iterativně.

Důležitým prvkem při začátku tvorby aplikace je její konfigurace. Aplikace začíná čtením souboru web.xml. Konfigurace je provedena jednou a to na začátku a v praxi je prováděna nejčastěji software architektem, který se stará mj. o kompatibilitu jednotlivých knihoven. V souboru ApplicationContext.xml jsou zavedeny další konfigurační soubory jako konfigurace databáze a webových toků. V rámci konfigurace databáze je obsaženo nastavení SŘBD pro poskytnutí URL adresy, uživatelského jména a hesla, dále spojení aplikace s Hibernate a řešení transakcí. U webových toků se jedná o nakonfigurování perzistence během provádění webflow, jejich ladění, a propojení JSF s Facelets (viz Příloha č. 5 - Konfigurační schéma beanů webových toků).

Dalším krokem je vytvoření Webflow Definition file, který určuje toky aplikace. S vytvořením toku souvisí i vytvoření webové stránky, která je typu XHTML s JSF tagy. XHTML je konvence pro JSF a Spring Webflow, a proto je doporučeno ji používat. Je vhodné vytvořit šablonu (angl. template) tedy webovou stránku, která bude mít základní prvky, které se na všech stránkách opakují. Technologie použité v šabloně a ostatních pohledech jsou XHTML, JSF, JSTL, Facelets a PrimeFaces.

Průběžné ukládání jednotlivých verzí aplikací je v dnešní době nutností téměř v každém projektu, ale rovněž i v rámci následování metodiky Agile Unified Process. V případě této práce je v rámci programu Eclipse využito modulu Subclipse, který umožnuje integraci se Subversion (SVN), systémem pro správu verzí aplikací. Pro správnou funkčnost tohoto pluginu je nutno nainstalovat do programu Eclipse tři moduly – Subclipse, Optional JNA Library a Core SVNKit Library.

Pro zajištění nejlepšího výkonu pluginu Subclipse je nutno nastavit v jeho vlastnostech rozhraní „SVNKit“ oproti původně zvolenému JavaHL (JNI). Dalším krokem je nalezení vhodného online nebo off-line úložiště pro projekt (angl. „project repository“). K dispozici je velké množství uložišť. Uložiště s kvalitní zákaznickou podporou a velikostí jsou zpoplatněné. Na druhou stranu existují i bezplatné úložiště, jenže mají omezený prostor pro kód aplikace nebo jsou omezeny pouze na určitý počet dní. Pro aplikaci bylo zvoleno úložiště „Project Locker“, které nabízí zdarma pro verzování aplikace prostor 50 MB pouze pro jednoho uživatele, ale časově neomezený.[3] Po zaregistrování je uživateli poskytnuta URL adresa projektového úložiště, včetně uživatelského jména a hesla. Tyto údaje je potřeba vyplnit v programu Eclipse pro připojení k úložišti.

Po propojení s projektovým úložištěm je aplikace verzována každým jejím nahráním. Před nahráním aplikace je potřeba nastavit různé parametry. Kromě URL adresy, uživatelského jména a hesla, je potřeba zadat ještě komentář k nahrávané verzi. Navíc je vhodné nenahrávat soubory, které nejsou potřebné pro běh aplikace jako konfigurační soubory a nastavení Eclipse, či další soubory spojené s vývojem aplikace v tomto vývojovém prostředí. Aby se zamezilo nahrávání těmto souborům, lze změnit výchozí nastavení SVN Eclipse a tyto soubory ignorovat.

4.1.1 Správa chyb

Pro správu jakýchkoli chyb v aplikaci je nutno zaprvé vytvořit nový webový tok pro zobrazení chybové stránky. Tedy kromě definičního souboru webového toku je potřeba vytvořit i stránku dle normy XHTML, jelikož je tato norma definovaná v konfiguračním souboru webové aplikace pro zobrazování pohledů s technologií JSF.

Na této stránce se objevuje obecné hlášení s odpovědným obrázkem a nabídkou možností k pokračování. V této nabídce je možnost jít na poslední zobrazovanou stránku nebo přechod na domovskou stránku s novinkami. Pro docílení přesměrování v případě jakékoli obecné chyby, která může vzniknout, je potřeba toto chování konfigurovat na aplikační úrovni. Pokud by šlo pouze o odchytávání chyb a výjimek z webových toků, mohl by nastat problém s chybami vzniklými v Java třídách. Konfiguraci je nutno zapsat do souboru `web.xml` umístěného ve složce `WEB-INF`.

[3] Více informací na internetové adrese: https://signup.projectlocker.com/signup

```
1.  <error-page>
2.      <exception-type>java.lang.Throwable</exception-type>
3.      <location>/app/error</location>
4.  </error-page>
```

Obr. 4.1 Konfigurační kód pro přesměrování na chybovou stránku

V případě, že při užívání aplikace nastane chyba, tak nejen, že je uživatel přesměrován na chybovou stránku, ale rovněž je tato chyba zapsána na straně serveru do textového souboru pomocí logovací knihovny log4j. Toto platí od vyhození běžné výjimky až po nastání kritické chyby. Do souboru je zapsána zpráva, dále třída, která chybu vyhodila s konkrétním řádkem, datum a čas nastání chyby, kompletní trasování a informace o serveru.

Tyto logovací soubory jsou ponechávány na serveru a jednotlivé verze jsou dynamicky udržované rovněž touto zmiňovanou knihovnou. Jediným způsobem jejich smazání je ruční vymazání administrátorem, který jako jediný má přístupová práva do adresáře aplikace na serveru. Jelikož ale zabírají zanedbatelné místo na disku v poměru k ostatním souborům, není třeba je průběžně mazat. Na druhou stranu jsou velmi důležité a můžou být použitelné pro odstranění chyb z aplikace v rámci testovacího, zaváděcího období nebo případně i v produkčním stádiu. Velmi důležitým prvkem je rovněž psaní komentářů ke každé metodě a konfiguraci aplikace, jelikož při opravování chyb se je možno mnohem rychleji zorientovat v kódu a funkcionalitě.

4.1.2 Přístup k databázi

Pro užívání databáze je třeba začít instalací ORACLE databáze 11g express edition. Dále následuje vytvoření schématu a uživatele databáze pomocí SQL Plus v příkazovém řádku a propojení aplikace s databází pomocí datasource-config.xml. V tomto konfiguračním souboru kromě jiných konfigurací je nutno nastavit i správný Oracle Dialect, který se ovšem co určité období mění, proto si je třeba dát pozor, aby nebyl použít zastaralý dialekt.

V pohledu je využito JSF, PrimeFaces a JSTL. Vytvořené JavaBeany se propojují s pohledy v jejich programovém toku konfiguračního souboru skrze nadefinování položky var a následně lze tuto proměnnou používat pro všechny pohledy, stačí jen přidat vlastnost `model`. JavaBean musí implementovat třídu Serializable a vygenerovat sériové číslo.

V rámci anotace Hibernate lze používat tagy jako `@Table` pro určení tabulky v databázi nebo `@Column` pro určení sloupce v databázi. Pokud není určeno jinak, je brán výchozí název třídní entity jako název tabulky, která se nachází v databázi. Posléze v souboru

persistance.xml je nutno přidat tag `class`, jež Java EE aplikaci ukazuje, které JavaBeany jsou třídy určené k persistenci.

V další fázi je nutno vytvořit společnou třídu pro všechny entitní databázové třídy (BaseEntity), která je taky JavaBeanem a slouží pouze pro uchování obecného `Id`, který má automatický inkrementální přírůstek a slouží všem tabulkám. Proto tento JavaBean bude dále rozšiřitelný. V rámci Oracle databází se používá u automatického přírůstku anotace `@GeneratedValue` a např. při použití RDBMS MySQL by tato anotace byla `@AutoIncrement`. Na tento výjimečný JavaBean je použito anotace `@MappedSuperclass`, což dává Hibernate najevo, že tomuto JavaBeanu nemá tvořit žádnou tabulku, ale tento JavaBean bude použit pouze s dědičností.

Jedna z velmi důležitých částí je Data Access Object, jak už bylo zmíněno v teoretické části. S tím je spojená třída GenericDao, která je rozhraním pro všechny DAO, je rozšiřitelná i implementovatelná. Toto rozhraní obsahuje generický typ, který umožňuje přijímat jakýkoli entitní typ. Aby bylo možné používat DAO, je potřebné vytvořit abstraktní třídu GenericJpaDao, která poskytuje propojení Java Persistance API a Data Access Object. Třída je abstraktní, protože bude muset být ještě dále rozšiřitelná pro specifické potřeby. I ona má generický typ, který umožňuje práci s rozdílnými typy entit. V této třídě se nachází jak generická proměnná `persistentClass`, která obsahuje právě daný entitní typ, tak proměnná `entityManager`, která obsahuje instanci hibernate entity managera, která ale nemůže být veřejně přístupná, a proto je její getter nastavený na hodnotu protected. Díky tomu může entity managera získat pouze ta třída, která rozšiřuje danou abstraktní třídu. Nastavení entity managera musí být veřejné, jelikož se k němu bude přistupovat s persistance context anotací, proto aby když se bude aplikace spouštět, je potřebné určit, kde bude vložená hibernate manager implementace. A to je provedeno díky anotaci `@PersistenceContext`.

Pro vyvolání SQL dotazů se v databázové vrstvě používá Hibernate Structured Query Language. Jelikož pracujeme s entitními třídami a ne tabulkami, tak se v HSQL používají entity. Například pro persistenci uživatele do databáze je vhodné vytvořit interface UserService. Obecně pro všechny služby (services) by se měly vytvářet interface, jelikož je to programátorská konvence. Implementováním tohoto interface se dodržují pravidla a konvence objektově-orientovaného programování a je třeba pamatovat, že lze mít více implementací stejných služeb.

4.1.3 Služby aplikace

Pro implementace služeb je konvence vytvářet nový balíček, který je svým názvem podobný rozhraním služeb, ale za ním následuje ".impl" jako implementace daných rozhraní. V třídě, která implementuje dané rozhraní služby, se přepisují metody. Pro práci s databázovými entitami a jejich operacemi je nutno nadefinovat nové rozhraní, v případě uživatele "UserDao". V tomto rozhraní se implementují obecné metody práce s databázovou entitou, ale také definují nové metody konkrétně pro daný případ (zde to jsou metody pro práci s uživatelskou entitou). V tomto případě lze vytvořit metodu, která kontroluje, zda uživatel s daným e-mailem už existuje či nikoli. V rozhraní se pouze vytvoří obecná definice metody a třída, která implementuje toto rozhraní (v tomto případě UserJpaDao) už konkrétně definuje danou metoda.

Tato implementační třída UserJpaDao dědí obecné metody uložit, změnit, smazat, apod. z generické nadtřídy a přidává nové metody, například zdali uživatelské jméno už existuje nebo načtení uživatele z databáze podle zadaného e-mailu, které implementuje z třídy UserDao.

Zajímavou vlastností pro práci s Data Access Object je třída Assert ze Spring frameworku. Tato třída obsahuje statickou metodu `notNull`, která zjišťuje, zda daná proměnná není nulová. Její použití je vhodné, jelikož kdyby byla předána nulová hodnota HSQL příkazu, tak je možné, že by došlo k neočekávanému chování. Pokud je objekt, který je parametrem této metodu opravdu nulový („null"), pak je vyhozena výjimka `IllegalArgumentException` a aplikace bude zastavena.

V rámci specifikace metod v implementační třídě UserJpaDao není pouze využití statické třídy Spring frameworku Assert, ale i zadávání HSQL příkazů pro dosažení cíle dané metody. V rámci HSQL příkazu standardně nedefinujeme název entity napevno, ale název dostáváme z instance persistentní třídy z důvodu, jelikož se názvy tříd aplikace můžou měnit, a proto by takto napevno navolené názvy entit začaly vyhazovat výjimky.

V implementační třídě (službě) userServiceImpl se nachází implementační metoda pro vytvoření nového uživatele. V této třídě se vyskytuje řídící část aplikace. V tomto případě se jedná o ověření, zda je uživatel už zaveden v databázi a pokud ne, tak až pak se může registrovat a může se vytvořit nový záznam. Řídící část aplikace se nesmí nacházet v Data Access Object třídách, jelikož tyto třídy poskytují prostor pouze pro metody komunikující

výhradně s databází (získávání a ukládání dat). Jakákoli jiná řídící logika musí být přenesena do řídících tříd.

Další částí je konfigurace závislostí (dependencies) protože proměnné DAO použité v řídících třídách (services) nejsou samy o sobě inicializované. Proto je nutno tyto proměnné inicializovat v konfiguračním souboru Spring Frameworku `applicationContext.xml`. V tomto konfiguračním souboru nadefinujeme Java Beany s parametry `id` a `class`. Je dobrým zvykem aby `id` bylo shodné s názvem DAO interface a do parametru `class` se vkládá implementační třída DAO. Dále je nutné vytvořit podobným způsobem bean pro řídící třídy. Jelikož se v řídící třídě nachází proměnná DAO, je nutné ji v kontextu aplikace inicializovat. Tato inicializace využívá vlastnosti Spring frameworku "dependency injection" tím, že vloží instanci dané třídy do specifické proměnné.

4.1.4 Konfigurace transakcí

V Hibernate není automatický nastavený příkaz `commit` a to znamená, že buď je nutno jej volat u každé metody, která nějakým způsobem ovlivňuje záznam v databázi nebo jsou použity anotace pro automatický příkaz `commit` při každém takovémto úkonu. Springem řízené transakce spouští příkaz `commit` při exit události dané metody.

Implementace Java Persistent API (JPA) mají možnost se starat o transakce samostatně (RESOURCE_LOCAL) nebo přenechají správu transakcí implementacím aplikačního serveru Java Transaction API (JTA).

Ve většině případů stačí samostatná správa o transakce (RESOURCE_LOCAL). Je tomu tak i v případě této řešené aplikace. Tento přístup využívá základní úroveň JDBC transakcí. Nevýhodou může být to, že transakce jsou lokální pro JPA persistentní jednotku, takže když by bylo zadáno, aby transakce zahrnovala více persistentních jednotek (nebo více databází), pak už by toto řešení nestačilo. V tomto případě je ale lokální správa řešení transakcí plně dostačující požadavkům aplikace.

Na druhou stranu je JTA kromě globálních aplikací používána pro správu transakcí mezi systémy jako například JMS (Java Messaging Services) a JCA (Java EE Connector Architecture), ale toto použití je v praxi vzácné. Pro použití JTA je nutná podpora na straně aplikačního serveru a taktéž podpora od JDBC ovladače.

4.1.5 Omezení přístupu

Jelikož je vyžadováno, aby přístup k některým stránkám byl omezen (např. na základě uživatelského jména), tak je nutné, omezit přístup na daný tok programu. Toto omezení se prování jednou z částí Spring Frameworku a to je Security. V minulosti se Spring Security nazývalo ACG Security. Jelikož Spring security poskytuje vývojáři bezpečnostní vrstvu, je vhodné ji v tomto případě využít. Na začátku zabezpečovací fáze je důležité nakonfigurovat bezpečnostní filtr jako Spring Security Filter Chain, který je načtený při načtení aplikace spolu s URL mapováním. Další důležitou částí je přidání WebFlow Execution Listener, ale ještě předtím je nutností definovat patřičný bean pro zabezpečení webového toku. Pro konfiguraci Spring Security je potřeba vytvořit konfigurační XML soubor. V tomto souboru je potřeba nastavit přesměrování při přihlášení uživatele a při odhlášení uživatele, správné kódování hesla, možnost vlastního odchycení výjimky týkající se neexistujícího uživatele v databázi a vytvořit bean pro správu autentizace.

Pro zabezpečení daného programového toku je nutno přidat v konfiguračním XML souboru daného toku značku `secured` s parametrem `attributes`, který obsahuje role uživatelů, kteří k danému toku můžou přistupovat. Pro přístup k zabezpečenému programovému toku, je potřeba prvně uživatele autentizovat a nastavit spring security nositelský soubor, který vytvoří session. Při vytváření hesla je důležité zvolit hashovací algoritmus. Jako jedny z nejbezpečnějších algoritmů implementovaných ve spring frameworku lze řadit algoritmus BCrypt, který má pouze dynamickou kryptovací sůl. V tomto projektu je využita právě tato hashovací metoda. Ostatní algoritmy (MD5, SHA-1, SHA-256, SHA-512, SHA-3 apod.) jsou navrženy k počítání obrovského množství dat v co nejkratším čase. To vede k tomu, že s rostoucím výpočetním výkonem je možno zkusit zadat mnohem více hesel ve stejném časovém rozmezí, díky čemuž vzniká velké nebezpečí díky tzv. útoku hrubou silou (angl. brute force attack). BCrypt je oproti předchozím zmíněným o dost pomalejším algoritmem, který zpomaluje proces autorizace uživatele, a díky tomu je poskytnuta i vyšší úroveň zabezpečení například proti útoku hrubou silou.

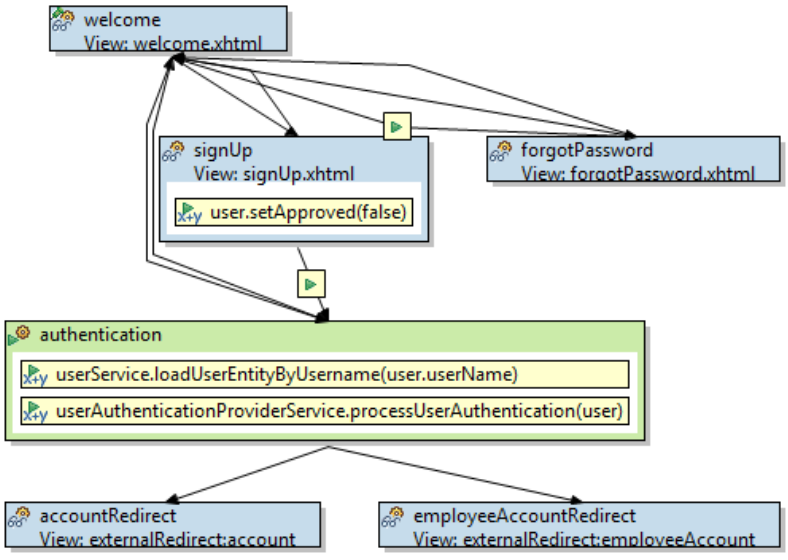
Implementace uživatelských rolí začíná vytvořením vztahu více k více mezi tabulkou uživatelů a rolí, protože uživatel může vlastnit více rolí a role může být přiřazena více uživatelům. Následně je potřeba nadefinovat entitní třídu pro tabulku uživatelských rolí. Při ověřování uživatelského jména a hesla musí být načtené uživatelské role s příslušné tabulky a vložené do Spring Security kolekce `GrantedAuthority`. Tato kolekce udržuje

všechny role přihlášeného uživatele, jelikož je možno mít více uživatelských rolí zároveň. Pokud uživatel aplikace nemá dostatečné oprávnění pro návštěvu nějakého webového toku, pak je přesměrován na chybovou stránku. To je docíleno vložením konfigurace pro přesměrování do souboru `web.xml`, pokud protokol HTTP zahlásí kód stavu 403.

Pro zabezpečení obsahu, který je pouze pro autorizované uživatele, je nutno zapsat konfiguraci do souboru security-config.xml, která vypne ukládání do vyrovnávací paměti. Tímto se zamezí nebezpečí, které by mohlo vzniknout při použití běžné funkce „zpět“ v prohlížeči. Při použití této funkce by byla dostupná i po odhlášení stránka, která vyžaduje zvláštní oprávnění.

Pro použití java beanu v programovém toku je nutno inicializovat tento bean v applicationContext (konfiguračním souboru Spring frameworku), a díky vlastnosti "dependency injection" vložit instanci cizího objektu do proměnné daného beanu.

Pokud během přihlašování do aplikace uživatel nebyl v tabulce uživatelů nalezen, je vyhozena výjimka "UsernameNotFoundException", které je předána chybová zpráva a celá výjimka je odchycena ve výjimce "AuthenticationException" během metody sloužící k autentizaci uživatele, při které je evokován informační dialog zobrazující chybovou hlášku. Tento dialog je implementován v knihovně JavaServer Faces. V aplikaci, při jejím načtení, je načten programový tok, ve kterém se rozhoduje, zdali je uživatel přihlášen a je dále přesměrován nebo přihlášen není a je přesměrován na uvítací programový tok "welcome", kde jsou požadovány buďto přihlašovací údaje nebo registrace nového uživatele.



Obr. 4.2 Schéma úvodního webového toku

Pro odesílání informačních, ale i výstražných či fatálních zpráv ze servisní vrstvy na konkrétní pohled, je nutno vytvořit instanci třídy "FacesMessages". V případě, že z jedné servisní třídy by bylo odesíláno větší než obvyklé množství zpráv (jako např. v uživatelské servisní třídě), je možno vytvořit pomocné metody, které budou automaticky instance vytvářet, a tím se jak zpřehlední kód, tak i zjednoduší.

4.1.6 Funkce a možnosti tabulek

Pro vytvoření moderní dynamické aplikace je nutno zapojit technologii AJAX do běhu programu. To je v tomto případě provedeno úpravou stávajících pohledů využitím knihovny PrimeFaces. Více dynamická je aplikace proto, že na událostí stisku klávesy je navázáno spuštění událostí. V případě registrace nového uživatele se při každém stisknutí klávesy kontroluje správnost vyplňovaného pole. Výjimku tvoří v případě pole "E-mail", které navíc při každém stisku klávesy vyvolává databázový dotaz na to, jestli je daná e-mailová adresa už registrovaná, či nikoli. Proto se musí nastavit ajaxový posluchač, který na dané události reaguje. Při implementaci metody, která je vyvolávána ajaxem je potřeba použít parametr "AjaxBehaviorEvent". Dalším krokem je inicializace komponenty, která dané volání vyvolala, a to lze zjistit právě z parametru této metody. Následně je získána textová hodnota z dané komponenty. A na konci ajaxem volané metody je poslána zpráva o dostupnosti nebo nedostupnosti uživatelského jména.

```
1.  public boolean checkAvailable(AjaxBehaviorEvent event) {
2.      InputText inputText = (InputText) event.getSource();
3.      String value = (String) inputText.getValue();
4.      boolean available = userDao.checkAvailable(value);
5.      if (!available) {
6.          FacesMessage message = constructErrorMessage(null,
                  String.format(getMessageBundle()
                  .getString("userExistMsg"), value));
7.          getFacesContext().addMessage(event.getComponent()
                  .getClientId(), message);
8.      } else {
9.          FacesMessage message = constructInfoMessage(null,
                  String.format(getMessageBundle()
                  .getString("userAvailable"), value));
10.         getFacesContext().addMessage(event.getComponent()
                  .getClientId(), message);
11.     }
12.
13.     return available;
14. }
```

Obr. 4.3 Kód pro zjištění dostupnosti uživatelského jména

Tabulky jsou tvořené komponentou datové tabulky („DataTable“) z knihovny PrimeFaces. Pro zobrazení všech záznamů uživatelů, v této aplikace pouze pro administrátorské účely, je zapotřebí volat v patřičné řídící třídě metodu, která získá záznamy z databázové vrstvy. V případě zobrazení všech záznamů se tedy v této metodě volá generickou třídu, která obsahuje metodu pro zobrazení všech záznamů (`findAll`). Tato metoda volá rovnou entitního manažera, který se už postará o dodání záznamů z databázové tabulky v systému řízení báze dat. Ten vrací záznamy entitního typu, které jsou postupně předávány přes servisní třídu až do PrimeFaces komponenty v uživatelském pohledu. Posléze se díky hodnotovému atributu mapuje výsledek servisní třídy s pomocí využití JSF výrazového jazyka dané komponentě a díky atributu `var` je možno přistupovat k jednotlivým instancím entitních tříd a využívat jejich vnitřních hodnot. Tyto hodnoty jsou mapované k jednotlivým sloupcům tabulky stejnou metodou a to výrazovým jazykem.

Řazení tabulky probíhá díky volání ajaxových metod uvnitř komponenty datové tabulky. V každém sloupci je zapotřebí nastavit atribut „sortBy“, který určuje, podle čeho se bude daný sloupec řadit. V rámci českého jazyka se řazení komplikuje. České písmena jsou ve výchozím stavu komponenty řazeny až na konec tabulky. Jako příklad je možno vzít jakékoliv slova začínající na písmena š, č, ř a ž. Toto chování je samozřejmě nevyžádané, jelikož je zákazník zvyklý na standardní řazení písmen a tedy za písmenem „c“ by mělo být „č“, za písmenem „r“ by mělo být „ř“ apod. Správného chování je docíleno vytvořením nové řadící metody, která přepisuje výchozí řadící metodu komponenty. Tato metoda obsahuje dva parametry, dva vstupní řetězce, které jsou porovnávány a následně řazeny mezi sebou. Vrácený výsledek je celočíselného datového typu, který vyobrazuje, jestli první řetězec má být před druhým či naopak. V metodě je vytvořená instance třídy „Collator“, do které je předán jako parametr instance třídy „Locale“, která slouží pro aplikování metod národního prostředí. Třída „Collator“ slouží pro porovnání dvou řetězců na základě patřičného národního porovnání. V metodě řazení se ještě na instanci třídy „Collator“ nastaví síla porovnávání na konstantní hodnotu „Identical“. Toto nastavení informuje danou instanci o tom, do jaké míry jsou řetězce podobné na základě unicode tabulky.

Pro filtrování záznamů tabulky se opět využívají vlastnosti datové tabulky z knihovny PrimeFaces, ovšem podpořené v servisní třídě aplikační logiky programu. Pro filtrování se používají značky „filteredValue” pro celou tabulku, ve které je výrazový jazyk a „filterBy“ pro jednotlivé sloupce. V servisní třídě je pak proměnná seznamu, která dočasně udržuje všechny filtrované výsledky. Tyto filtrované výsledky jsou pak předány komponentě datové

tabulky, která je následně zobrazí. Opět celá komunikace probíhá jak ajaxovým voláním, tak i klasickým požadavkovým v případě, že je JavaScript vypnut.

Stránkování tabulky je nastaveno ve výchozím stavu na deset záznamů na stránku. Lze si vybrat z možnosti pěti nebo patnácti záznamů. Navíc lze procházet mezi jednotlivými stránkami, ale rovněž i skočit na poslední nebo naopak na první stránku. V patičce tabulky je taktéž zobrazena informace s číslem právě zobrazené stránky a celkový počet stránek. I tato funkce využívá hlavně ajaxových volání, ale není to nutností. Slouží pouze jako doplněk.

Každou tabulku aplikace lze snadným způsobem exportovat do dvou formátů. Jedním je formát programu Microsoft Excel s příponou XLSX. Dalším formátem je export celé tabulky do formátu PDF. Oba formáty plně podporují české znaky a české řazení znaků. Po exportu je tedy velice snadné všechny záznamy vytisknout bez nutnosti tisknout webovou stránku jako celek. Exportní funkce využívá metod knihovny „PrimeFaces Extensions“, která musí být zanotovaná na začátku webové stránky v XML anotaci, stejně jako ostatní knihovny využívané v aplikaci, případně na dané stránce.

4.1.7 Přidání, změna a smazání záznamů

V případě přidávání uživatelů je postup zřejmý, jelikož je podobný jako v ostatních případech přidání záznamů do tabulky. Jako první krok je potřeba vytvořit stavu pohledu, kterému je přidělena vlastní webová stránka. Na této stránce se díky JSF komponentám vytvoří přehledný formulář podobný registračnímu formuláři. K tomuto formuláři pro přidání uživatelů má přístup pouze administrátor. V tomto formuláři, oproti formuláři vlastní registrace uživatelů je rozdíl v tom, že po zaregistrování uživatele administrátorem, je tento uživatel ihned automatický schválen. Naopak v případě samovolné registrace uživatelů je uživatel ve výchozím stavu po zaregistrování neschválen a musí být schválen administrátorem. Toto schválení probíhá jednoduchým stiskem na tlačítko „Schválení“ v tabulce uživatelů s následným zasláním e-mailu uživateli o jeho schválení. Po stisknutí je uživatel schválen a může využívat aplikaci. Pokud ne, nemůže se do aplikace přihlásit. V rámci aplikační logiky pro přidání uživatele se data z formuláře pošlou pomocí metody POST do stavu pohledu, následně do programového toku, který je přepošle metodou servisní třídy do generické třídy, která volá metodu `persist` na instanci entitního manažera.

Změna uživatelových dat je dostupná příslušnému uživateli a administrátorovi aplikace. Aplikační logika změny uživatele je velmi podobná procesu přidání uživatele jenom s rozdílem, že na instanci entitního manažera se volá metoda `merge`. Editace uživatele probíhá

po stisku ikony tužky vpravo od řádku v tabulce. Po stisku jsou textové řetězce změněny na vstupy v rámci celého řádku a ikona tužky pro úpravu je změněna na ikonu potvrzení změn a ikonu křížku pro zrušení změn. Po úspěšné změně je zobrazena informační hláška se zprávou úspěšného změnění. V případě neúspěšné změny se zobrazí chybová hláška s výstražnou ikonou. Po potvrzení jsou změny zobrazeny ihned v pohledu na webové stránce, ale i v databázi.

V rámci provádění změn v databázové tabulce jsou u této metody jako u dalších metod nastaveny transakce, které jsou tvořeny za běhu aplikace přesně při zavolání metody používající databázi. Toto je umožněno díky Spring anotacemi řízené aplikaci, a právě anotace přiřazují transakce těmto metodám.

Smazání uživatele, je opět obdobné smazání řádku v jakékoli tabulce. Je ovšem mírně odlišné od přidání a změny řádku tabulky. Ještě v případě smazání uživatele, není nutno přímo uživatele mazat. Stačí, když bude uživatel nastaven jako neschválen a nebude se moct přihlásit do aplikace. Na druhou stranu je zřejmé, že úplné smazání uživatele je vhodné podle konvencí, ale i pro úplnou kontrolu nad databázovými daty administrátorem. Ovšem při úplném, trvalém smazání uživatele s tabulky proběhne smazání všech záznamů v navázaných relačních tabulkách. Proto je nutno se před smazáním uživatele dobře rozmyslet, zda funkce neschválení daného uživatele nestačí.

V samotné implementaci je pak při pokusu o smazání uživatele vytvořeno dialogové okno pomocí PrimeFaces komponenty „CommandLink“ a jejímu parametru `onclick`, které vyvolá dialogové okno s potvrzující otázkou, zdali administrátor opravdu chce smazat uživatele s příslušným jménem a příjmením. Rozhodne-li se programátor pro smazání uživatele, pak proběhne uživatelovo smazání opět ihned jak z pohledu, díky vlastnosti „action“, která aktivuje příslušnou akci v programovém toku aplikace, což vede v konečném důsledku k smazání řádku z databázové tabulky všech uživatelů.

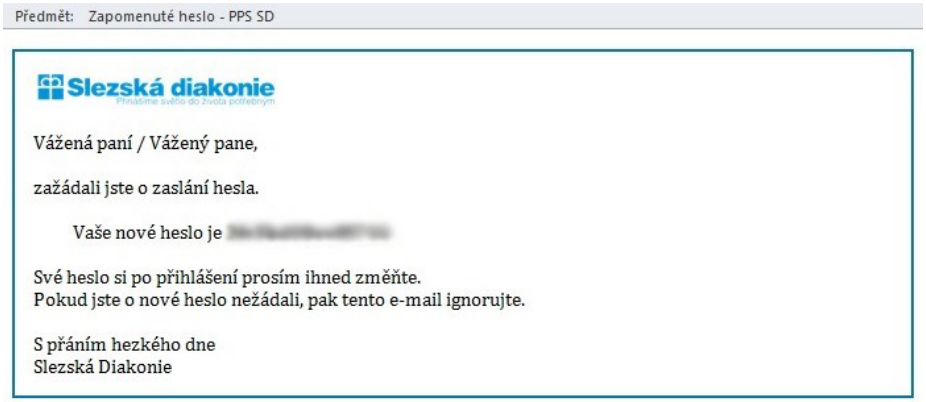
Počátek odstraňovací logiky uživatele z databáze je analogický s přidáním a úpravou uživatele, jen s rozdílem ověřování, zda entitní třída je podtřídou základní entitní třídy, která nese pouze identifikační parametr, opakující se ve všech entitních třídách, tedy zdali entitní třída obsahuje přírůstkový identifikátor, či nikoli. Podmínka využívá metody `class.isAssignableFrom` volanou na základní třídě. Tato metoda umožňuje rozpoznat datový typ parametru. Je v pořádku používání i metody `isInstance`, ale její použití je vhodné pouze tehdy, pokud se pracuje pouze s referenčními datovými typy. Ale pokud jde naopak

o primitivní datové typy a je použita metoda „isInstance“, pak můžou být vráceny neočekávané výsledky. Proto je použití metody „class.isAssignableFrom“ bezpečnější a vhodnější v generických třídách.

Tato podmínka využívá poměrně nové vlastnosti programovacího jazyka Java a konkrétně Reflexe. Díky ní, je možno volat, tvořit, měnit třídy a např. metody aplikace až po jejím zkompilování v rámci běhu programu (runtimu). Pokud je podmínka splněna je na instanci entitního manažera volána metoda `remove` s referenčním identifikačním parametrem základní třídy entitního typu. V případě, že podmínka splněná není a tedy aplikace pozná, že tabulka nemá automaticky generovaný přírůstkový identifikátor, zavolá na instanci entitního manažera metodu `merge` a následně `remove` (viz Příloha č. 6 - Diagram tříd pro správu uživatele aplikace).

4.1.8 Zapomenuté heslo

Pokud uživatel při pokusu o přihlášení zjistí, že zapomněl heslo, existuje v aplikaci funkce zaslání nově vygenerovaného hesla na e-mail. Formulář pro zadání e-mailové adresy se nachází na samostatné webové stránce s vlastním stavem pohledu. Po odeslání formuláře se e-mailová adresa použije jako parametr metody v servisní třídě. V té se zjišťuje, zdali e-mailová adresa je opravdu zaregistrovaná v aplikaci. Pokud ne, je vyhozená chybová hláška. Pokud ano, pak je načtená celá entita do proměnné pro snadnou změnu hesla. Heslo je automaticky vygenerováno pomocí HEX kódu, pak je uloženo do entity a změněno v databázi. Následně proběhne zaslání hesla na e-mailovou adresu spolu s výzvou ke změně hesla ihned při příštím přihlášení. K zaslání e-mailové zprávy se využívá součást Spring frameworku knihovna „JavaMail“, díky které lze nastavit např. SMTP server, či port a v rámci psaní zprávy je umožněno využít HTML jazyka včetně zaslání přílohy spolu s danou e-mailovou zprávou.



Obr. 4.4 Ukázkový e-mail zaslaný při zapomenutí hesla

Výše je uvedená ukázka e-mailu zaslaného uživateli aplikace, který zapomněl své heslo. V této zprávě je využito jazyka HTML spolu s přidáním obrázku, konkrétně loga organizace. Obsah e-mailové zprávy je generovaný dynamicky pro uživatele, který o heslo zažádal, obsahující bezpečností kód a informace o změně hesla po přihlášení.

4.1.9 Správa smluv a novinek

Uživatel aplikace a tedy zaměstnanec organizace si smí vybrat při žádosti o připojení k privátní podnikové síti zda chce dostat nové číslo nebo chce pouze převést svoje stávající číslo od operátora T-Mobile případně nějakého jiného operátora. Tato volba ovlivní, do které databázové tabulky bude přidán záznam. Jelikož už je díky fázi návrhu známo, že tabulky zahrnující seznam smluv jsou rozděleny podle generalizace, tedy na jednu nadřazenou tabulku a tři podřazené tabulky dědící všechny vlastnosti nadřazené tabulky.

Jelikož je využito objektově relační řešení mapování jedné třídy k jedné tabulce v rámci generalizace tříd, a to ať už jde o nadtřídu nebo podtřídu, musí primární klíče tabulek mapovaných jako podtřídy být rovněž cizími klíči k primárnímu klíči nadřazené tabulky. V takovém případě je využito konfigurace připojených podtříd. Jde tedy o tabulku „Smlouva“ (Contract) sloužící jako nadtřída a podtřídy jsou tabulky „Převod telefonního čísla“ (Transfer_of_number), „Účastnická smlouva LE“ (Le_collective_contract) a „Převod účastnických smluv” (Transfer_of_contract).

Ve výsledném řešení je zahrnuta jedna abstraktní entitní třída definovaná jako nadtřída spolu s anotací `@Inheritance` a tři podtřídy rozšiřující nadtřídu s využitím anotací `@PrimaryKeyJoinColumn`, ve které je definován sloupec s primárním klíčem pro spojení tříd (viz Příloha č. 7 – Ukázkový kód entitních tříd s dědičností). Základní třída, která tvoří nadtřídu pro ostatní podtřídy, je definována jako abstraktní a to je proto, aby nebylo možno vytvářet její instance, ale pouze instance jejich podtříd. Tím se zabezpečí potřebná integrita databáze.

Pro generování smluv, tedy naplňování polí v softwaru Microsoft Word daty z databáze, se nabízí tři možnosti. Prvním řešením je nahrazení proměnných, při němž dochází k nalezení daných proměnných ve struktuře dokumentu a následně jejich nahrazení textovými řetězci. Tyto proměnné by měly být ve tvaru `${key1}`, `${key2}` apod. Tento přístup lze využít pouze v jednoduchých případech, protože v ostatních případech, tj. když se klíč ovládacího prvku vyskytuje v dokumentu víc než jednou, je toto řešení nevhodné. Druhou možností je vázání ovládacích prvků obsahu na části XML (přes „XPath“).

V tomto případě je nutno všechny šablony převést na požadovaný tvar. Poslední možností a ta je v tomto projektu zvolena je použití „Merge Fields“ (tedy polí hromadné korespondence).

Pole hromadné korespondence jsou určeny pro nahrazení prostým textem. Vložení takovéhoto pole do dokumentu Word se provádí otevřením karty „Vložení“ v horním menu, dále „Rychlé části“ a „Pole“. V levé nabídce názvu polí je nutno zvolit „MergeField“ a nastavit název tohoto pole. To po vložení do dokumentu je složeno z kódu pole, které určuje jeho formátování, jež je skryto před uživatelem a výsledku pole, které obsahuje poslední vloženou hodnotu. Tato druhá část je už uživateli zobrazena. Celé pole pak může vypadat např. `{ MERGEFIELD Adresa /*MERGEFORMAT | «Adresa» }`. Jelikož je vkládání přes „Rychlé části“ pomalé, je možno využít rovněž možnosti, kdy lze převést prostý text na pole hromadné korespondence při použití klávesové zkratky `CTRL + F9`. Takto převedený text na pole musí být ale ještě aktualizován, aby vypadalo jako běžné pole hromadné korespondence.

Vkládání textových řetězců do polí v jazyce Java je umožněno open source knihovnou docx4j. Jako první krok je nutno vytvořit instanci objektu `WordprocessingMLPackage`, které parametrem je zpracovávaný dokument Word. Proces nahrazení polí probíhá v metodě `performMerge()` třídy `MailMerger`. Parametry této metody tvoří instance dříve používané třídy s původním dokumentem a asociativní pole (v tomto případě `HashMap`) obsahující názvy polí hromadné korespondence jako klíče a hodnoty, které mají být do těchto polí vloženy. Na instanci třídy s původním dokumentem je pak nutno zavolat metodu `save()`, která uloží dokument do složky s požadovaným názvem.

Po vygenerování smlouvy do formátu DOCX je požadavkem aplikace poskytnout taktéž formát PDF pro uživatele, kteří nemusí mít koupenou licenci Microsoft Word. Tento převod je už ale značně jednodušší než v minulém případě a je ho možno docílit rovněž využitím knihovny docx4j. Převod dokumentu s příponou DOCX do dokumentu PDF je umožněno díky třídě `Conversion` z balíčku `org.docx4j.convert.out.pdf.viaXSLFO`. U té je nutno zavolat metodu `output()`, které lze nastavit cestu výstupního PDF souboru a jeho konfigurační nastavení.

Po vygenerování smlouvy, jejím zpřístupnění v systému a zaslání na e-mail zaměstnance je nutno danou smlouvu podepsat a nahrát zpět do systému pro kontrolu administrátorem. Nahrání smlouvy do aplikace je umožněno komponentou „FileUpload“ z knihovny PrimeFaces.

Pomocí této komponenty je umožněno nahrání smlouvy zpět do systému prostřednictvím přehledného formuláře. Díky ní je poskytnuta podpora omezení přípon souborů, jejich maximální velikost a počet, ale i samostatná správa nahraných souborů.

```
1.  <ui:define name="content">
2.     <h:form name="uploadForm">
3.        <p:fieldset styleClass="fieldset"
                      legend="Nahrání smlouvy">
             <p:fileUpload fileUploadListener=
                "#{contractService.handleFileUpload}"
                allowTypes="/(\.|\/)(bmp|gif|jpe?g|png|tiff|pdf)"
5.              description="Vyberte soubory naskenové smlouvy"
                sizeLimit="20000000" update="msgs"
                dragDropSupport="true" multiple="true"
                fileLimit="10" />
7.        </p:fieldset>
8.     </h:form>
9.  </ui:define>
```

Obr. 4.5 Kód v pohledu pro vkládání naskenovaných smluv

Výše uvedená konfigurace komponenty „FileUpload“ umožňuje nahrání více souborů najednou, pokud by uživatel chtěl hromadně nahrávat jednotlivé naskenované stránky, soubory musí mít přípony jako obrázky případně PDF, jelikož jiné formáty nejsou potřeba pro naskenované soubory, limit velikosti nahraných souborů je 20MB, komponenta podporuje vybrání souboru přetažením do těla komponenty a maximální počet souborů je deset.

Součástí tvořeného systému je rovněž správa novinek. Tyto novinky má oprávnění vkládat pouze administrátor aplikace. Slouží hlavně pro informování uživatelů o nových funkcích v systému. Může obsahovat i rady a postupy pro správné zacházení nebo jednoduché sdělování potřebných informací.

4.1.10 Lokalizace

Internacionalizace aplikace, tedy zpřístupnění aplikace pro koncové uživatele dorozumívajícími se různými jazyky je dnes již nutností pro systémy, ke kterým přistupují nebo můžou přistupovat osoby z různých zemí. Proto je i v tomto projektu využito řešení pro vícejazyčnost aplikace a velmi snadným způsobem lze přidat jakékoli jazykové mutace.

Optimálně by všechny hlášky aplikace byly uloženy v jednom souboru ve tvaru „`*.properties`”. Takovýchto souborů je pak tedy tolik, kolik je jazykových alternativ potřeba. Toto řešení je velmi vhodné pro použití, protože stačí vytvořit jeden soubor ve známém jazyce, např. angličtině a pro překlad do jiných jazyků stačí poslat překladatelům

pouze tento jeden soubor. Je to veliká výhoda, protože není třeba odhalovat žádné části kódu, zdlouhavě je hledat, nahrazovat textové řetězce pro každý jazyk zvlášť apod.

Jednotlivé zprávy či hlášky se získají ze zdrojového souboru pomocí třídy `ResourceBundle` metodou `getBundle(String název)`, které parametrem je název souboru. Tímto jsou načteny do paměti všechny uložené zprávy. Konkrétní text je pak získán pomocí metody `getString(String název_proměnné)`. Do těchto zpráv lze vkládat i obsah běžných proměnných pomocí metody `String.format()`, kdy do závorek lze napsat jako první parametr textový řetězec obsahující značky např. ‚`%s`' a druhým parametrem je proměnná, která má být vložena na její místo.

4.2 Zabezpečení systému

Základem zabezpečení aplikace proti útokům je autorizace a autentizace uživatelů. Této možnosti poskytnutné díky Spring Security je už využito a je pomocí ní zabezpečen přístup na webové toky. Je možno taktéž zabezpečit jednotlivé metody aplikace pro různé uživatelské role. Šifrování hesla je nastaveno na BCrypt, které znemožňuje útoky hrubou sílou nebo slovníkové útoky, protože značně zpomaluje proces ověřování hesla a tím řeší problém zvyšujícího se počtu pokusů prolomení hesla se zvyšující se výpočetní silou. Druhým důležitým milníkem je vložení různých bezpečnostních hlaviček jako součást odpovědi serveru. Spring Framework poskytuje jejich jednoduché vložení do systému pouze pomocí vložení značky `<headers />` do konfiguračního XML souboru.

Ve výchozím stavu mezi tyto hlavičky počítáme nastavení správu mezipaměti, při jejímž nastavení je zajištěno, že prohlížeč nemůže ukládat zabezpečené stránky do mezipaměti. Dále je zde možno počítat zamezení prohlížeči zjišťovat typ obsahu odpovědi, čímž je poskytnuta ochrana před soubory s více typy obsahu, pomocí nichž lze spouštět XSS útoky. Jednou z hlaviček, které lze měnit jsou možnosti rámců, při jejichž zakázání není možno vkládat stránky systému jako rámce do jiných stránek. Při umožnění vkládání stránek do rámců jiných aplikaci je umožněno útočníkovi pomocí jednoduchých CSS příkazů podstrčit falešné tlačítko, kde se tváří například jako přihlášení do aplikace, ale přitom poskytuje přístup do systému i jiným uživatelům. Tato invektiva se nazývá útok založený na rámech (angl. clickjacking). Posledním zabezpečením v rámci vkládání bezpečnostní hlavičky do odpovědi serveru je základní ochrana proti XSS útokům. Tyto útoky jsou vyvolány vložením spustitelného kódu do hlavičky odpovědi, přičemž útok není uložen v aplikačním kódu, není trvalý a ovlivňuje pouze ty uživatele, kteří otevřou upravený odkaz

nebo webovou stránku třetí strany. Podstatou zabezpečení je filtrování těchto útoků. Tedy vložení bezpečností hlavičky zajistí, že filtrování je nastaveno a je provedeno zablokování nebezpečného obsahu spíše než snaha o jeho opravení, které může být rovněž nebezpečné. V aplikaci je využito všech nastavení hlaviček s výchozím doporučeným nastavením.

Zabránění „SQL Injection“ útokům je docíleno psaním bezpečných příkazů sql a tedy tím, že jsou parametry dotazu vkládány pomocí metody `setParameter()`, jež poskytuje entitní manažer aplikace. Tímto jsou všechny znaky, které můžou znamenat potenciální ohrožení kódovány do zabezpečené podoby.

Velkým nebezpečím aplikace je „Cross-Site Request Forgery”, a proto účinná ochrana před tímto útokem má vysokou prioritu. Útok je založen na vykonání akcí útočníkem pod identitou jiného uživatele (tedy oběti). Jelikož se ve tvořené aplikaci vyskytují citlivá data, je povinností se před tímto útokem bránit. Zabezpečení je umožněno díky už dříve zmiňovanému modulu Security z Spring Frameworku. V tomto případě je dostatečné vložit značku `<csrf />` do konfiguračního XML souboru a posléze je možno přidat do každého formuláře aplikace skryté pole, které se při každém vytvoření formuláře vygeneruje a s každým jeho odesláním je tento kód ověřen.

```
1.  <input type="hidden" name="${_csrf.parameterName}"
                    value="${_csrf.token}" />
```

Obr. 4.6 Kód skrytého pole pro ochranu přes CSRF útokem

„Session fixation“ je útok, který rovněž vyvolává potenciální riziko bezpečnosti aplikace. Útočník získá navštívením nějaké stránky session a následně ji podstrčí oběti, která se s touto session přihlásí do svého uživatelského účtu, v praxi např. pošle odkaz, ve kterém je session jako parametr. Spring Security modul zabraňuje takovémuto útoku automaticky tím, že vytváří novou session s každým přihlášením uživatele do systému.

4.3 Testování aplikace

Webovou aplikaci lze testovat pomocí manuálního nahlížení do výstupu jednotlivých metod. Tento proces se stává komplikovanější v případě, že se jedná o komplexnější aplikaci a zvláště pokud se provádí změny už manuálně testovaných metod a tříd. Po takové změně je nutno provést opět manuální vypisování vrácených hodnot z metod na konzoli, to se může opakovat velmi často a takovéto testování je naprosto neefektivní. Unit testy tyto problémy

řeší a navíc jednotkově testovaný kód lze logicky považovat za rychlý a spolehlivý. Velmi důležité je rovněž akceptační testování, kdy dochází k odsouhlasení aplikace zákazníkem.

4.3.1 Testování jednotek

Pro provádění testování jednotek je nutné do projektu přidat knihovnu JUnit pomocí nástroje Maven. To umožňuje kontrolovat správnou činnost metod, jejich zpracování a výstupy. Tyto testy je dobré provést u všech tříd, zejména pak u servisních tříd, které obsahuji procesní logiku aplikace. JUnity se píšou jako Java třídy do složky projektu `src/test/java`. Základem těchto testů ve Spring Frameworku jsou anotace `@Before` a `@Test`. První je anotace metody `setUp()` pro vykonání kódu před spuštěním testu, tedy vytvoření instance testované třídy, ale toto není nutné z důvodu umožněného vkládání závislostí a získání instance třídy přímo z `ApplicationContext`. Druhá anotace patří k testovací metodě, která ověřuje metodu testované třídy. Uvnitř je využito třídy `Assert` z balíčku `JUnit`. Metody, pro nastavení testu a provedení testu, jsou veřejné a nevrací žádnou hodnotu. Z třídy `Assert` jsou použity metody `assertEquals()`, která ověřuje, zda vrácená proměnná testované metody je shodná s očekávaným výstupem. Dalšími použitými alternativami jsou `assertNull()` a `assertNotNull()`, které ověřují, zda vrácená proměnná je nebo není nulová.

```
1.  @RunWith(SpringJUnit4ClassRunner.class)
2.  @ContextConfiguration(locations={"classpath:applicationContext.xml"})
3.  @Transactional
4.  @TransactionConfiguration(defaultRollback=true)
5.  public class UserServiceImplTest {
6.
7.          @Autowired
8.          private UserDao userDao;
9.
10.         @Test
11.         public void testGetUserList() {
12.                 assertNotNull(userDao);
13.
14.                 List<UserEntity> userList = userDao.findAll();
15.                 assertEquals(6, userList.size());
16.
17.                 UserEntity firstEntity = userList.get(0);
18.                 assertEquals("f@f.com", firstEntity.getUserName());
19.         }
20. }
```

Obr. 4.7 Unit Test metody getUserList třídy UserServiceImpl

Test uvedený výše testuje zda vložená závislost proměnné userDao není nulová, získává všechny uživatele aplikace a kontroluje, zda jejich počet odpovídá předpokládanému počtu, tedy šesti; a následně zjišťuje, zda první uživatel seznamu má uživatelské jméno "f@f.com". Tento test proběhl bez chyb a na vývojovém prostředí trvalo jeho vykonání 12,879 sekund. Ostatní metody servisních tříd taktéž proběhly bez chyby. Je důležité neplést si vykonání testovací metody na vývojovém prostředí s proběhnutím metody na produkčním prostředí, které má mnohem vyšší výpočetní sílu a tedy čas zpracování je pak mnohem kratší.

4.3.2 Akceptační testování

Důležitou části testování aplikace je taktéž akceptační testování. To provádí zákazník systému nejčastěji při použití uživatelských scénářů. V této fázi už samozřejmě nesmí být žádné základní chyby, které by uživateli znemožnily např. už jenom spuštění aplikace.

Jelikož je seznam uživatelských scénářů vypsán již ve fázi návrhu (viz Kapitola 3.3 Návrh nového systému), může zákazník aplikovat právě tyto postupy pro akceptační testování. V případě řešeného projektu je aplikace klientem průběžně testována při vydání nové verze podle dané iterace projektu. Verze systému, ve které se teď aplikace nachází, je zákazníkem schválená.

4.4 Zhodnocení výsledného řešení

Aplikace je stále ve vývoji proto nelze říct, že je vývoj a tím i projekt úspěšně dokončen, ale požadavky, které v průběhu vznikají, jsou postupně splňovány, řešení jsou testována a následuje pokročení k další verzi aplikace. Výsledné řešení je v nynější době zákazníkem přijato a schváleno, ale ještě není konečné. Není tedy přesně známo, kdy dojde k produkčnímu nasazení aplikace.

Při vývoji systému jsou použity moderní a osvědčené technologie jako Spring Framework s objektově relačním mapováním Hibernate, díky kterým je umožněno hlavně zabezpečení proti aktuálním, ale i starším hrozbám, optimalizace aplikace, využití nových funkcí komponent oproti jejím předešlým verzím apod. Jako systém řízení báze dat je zvolen produkt 11g Express Edition od Oracle a pro další projekty je možno ho doporučit, protože s ním nebyly žádné problémy a nevykazoval chyby. Problémem při realizaci webové aplikace bylo zajištění kompatibilních verzí knihoven. Některé knihovny nemůžou být použity v nejnovější verzi, protože nejsou zpětně kompatibilní s knihovnami, které už nejsou nadále vyvíjeny.

Pomocí evidenčního systému zákazníků privátní podnikové sítě je poskytnuta elektronická podpora jejich správě, příp. zpracování přehledů a smluv. Tento systém je rychlý, spolehlivý a do jisté míry interaktivní i s e-mailovým účtem v případě zaslání zapomenutého hesla a vygenerované smlouvy zaslané pro podepsání. Ukázky aplikace lze nalézt v příloze č. 8 – Náhledy vytvořené aplikace.

5 Závěr

Hlavním cílem diplomové práce byla analýza, návrh, implementace a testování systému evidence zákazníků pro privátní podnikovou síť neziskové organizace včetně splnění dílčích cílů zadaných zadavatelem, jako hlídání nutného prodloužení smlouvy, možnost změny, zrušení nebo zavedení nové smlouvy, automatické vyplňování a generování smluv napsaných v softwaru Microsoft Word s formulářovými prvky, které měly být nahrazené daty z databáze, autorizace uživatelů administrátorem, funkce vygenerování hesel všem zaměstnancům a zaslání na e-mail při zavádění aplikace, psaní novinek administrátorem nebo nahrávání podepsaných smluv do systému pro potvrzení a zavedení apod.

Práce se po úvodu zaměřuje na popis teoretických východisek, která je nutno pochopit pro následné pokračování. Do nich se řadí technologie pro realizaci aplikace, návrhové vzory a souhrn metodik vývoje softwaru s následnou volbou nejvhodnější varianty. Další částí je zpracování analýzy a návrhu softwaru, tedy stručný popis organizace, která je zadavatelem, analýza požadavků a soupis funkcí, procesů a vazeb v modelovaném systému. Rovněž jsou v této části vypsány uživatelské scénáře využívání aplikace. Následující kapitolou je realizace, zabezpečení a testování vyvíjeného softwaru. Není opomenuto ani zhodnocení celého systému včetně procesu tvorby. Souhrnně jsou tyto části naplněním všech bodů zadání diplomové práce.

Původní cíl i podcíle byly splněny, protože i když je systém ještě stále ve vývoji kvůli nově vzniklým funkcím, které nebyly známy na začátku, byla aplikace vytvořena a nachází se ve verzi, která splňuje hlavní požadavky zadavatele a je jím prohlášená za schválenou, i když ne ještě konečnou. Tohoto výsledku bylo dosaženo kombinací různých knihoven, které ve vzájemné kombinaci tvoří ucelený nástroj pro efektivní tvorbu webových aplikací. Velkým přínosem bylo rovněž využití agilní metodiky pro vývoj softwaru kvůli dostatečné spolupráci se zákazníkem, iteračnímu vývoji a reakci na změnu. Systém evidence zákazníků podnikové sítě, který je výsledkem této práce, je použitelný v reálném prostředí, a očekává se, že po zavedení do organizace zvedne efektivitu pracovníků majících na starost správu uživatelů privátní podnikové sítě z důvodu zautomatizování úkolů, které jsou povinni opakovaně provádět.

Seznam použité literatury

ASNANI, Satish. *Oracle Database 11g: Hands-on SQL and PL/SQL*. New Delhi: Motilal Uk Books Of India, 2010. 425 s. ISBN 978-81-203-4020-6.

BARNES, Joshua. *Implementing the IBM Rational Unified Process and solutions: a guide to improving your software development capability and maturity*. Upper Saddle River, NJ: IBM Press, 2007. 184 s. ISBN 978-032-1369-451.

BAUER, Christian a Gavin KING. *Java Persistence with Hibernate.* 2nd ed. Greenwich: Manning Publications, 2007. 408 s. ISBN 19-323-9488-5.

BUYYA, Rajkumar. *Object-oriented programming with Java: essentials and applications*. New Delhi: Tata McGraw-Hill, 2009. ISBN 978-007-0669-086.

CHOPRA, V., S. LI a J. M. GENENDER. *Professional Apache Tomcat 6*. Indianapolis: Wrox/Wiley Pub., 2007. 629 s. ISBN 978-0-471-75361-2.

CIMOLINI, Patrick a Karen CANNELL. *Agile Oracle application express*. New York: Apress, 2012. ISBN 978-143-0237-600.

DARWIN, Ian F. *Java cookbook*. 3rd ed. Sebastopol: O'Reilly Media, 2014. ISBN 978-144-9337-049.

DAVIS, Adam. *What's new in Java 8*: *An Unofficial Guide*. Charleston: CreateSpace Independent Publishing Platform, 2014. ISBN 978-1-4975-3350-9.

DEINUM, Marten et al. *Pro Spring MVC: with Web Flow*. New York: Springer Science Business Media, 2012. 565 s. ISBN 978-143-0241-560.

FAIN, Yakov. *Java programming 24-hour trainer*. 2nd ed. Indianapolis: Wrox/Wiley Pub., 2015. 624 s. ISBN 978-1-118-95145-3.

FAIRLEY, Richard E. *Managing and leading software projects*. Hoboken: Wiley, 2009. 492 s. ISBN 978-0-470-29455-0.

FISHER, Paul Tepper a Brian D. MURPHY. *Spring persistence with Hibernate*. New York: Apress, 2010. ISBN 978-143-0226-321.

FOWLER, Martin. *Destilované UML*. Praha: Grada, 2009. 173 s. ISBN 978-80-247-2062-3.

GEARY, David M. a Cay S. HORSTMANN. *Core JavaServer faces*. 4th ed. Upper Saddle River, NJ: Prentice Hall, 2015. 672 s. ISBN 978-0-13-173886-7.

GONCALVES, Antonio. *Beginning Java EE 7*. Berkley: APress, 2013. 608 s. ISBN 978-143-0246-268.

GREENWALD, R., R. STACKOWIAK a J. STERN. *Oracle essentials: Oracle database 11g*. 4th ed. Sebastopol: O'Reilly, 2008. 386 s. ISBN 978-059-6514-549.

HAAN, L. de, T. GORMAN, I. JØRGENSEN a M. CAFFREY. *Beginning Oracle SQL: for Oracle database 12c*. 3rd ed. New York: Apress, 2014. 440 s. ISBN 978-1-4302-6556-6.

HLAVATS, Ian. *Instant PrimeFaces starter*. Birmingham: Packt Pub., 2013. ISBN 978-1-84951-990-8.

HO, Clarence a Rob HARROP. *Pro Spring 3*. New York: Springer Science Business Media, 2012. 912 s. Expert's voice in Spring. ISBN 978-1-4302-4107-2.

JAGIELSKI, Piotr a Jakub NABRDALIK. *Instant Spring Security starter: learn the fundamentals of web authentication and authorization using Spring Security*. Birmingham: Packt Pub., 2013. ISBN 978-1-78216-883-6.

JENDROCK, Eric. *The Java EE 6 tutorial: advanced topics*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, 2010. 526 s. ISBN 978-0-13-708186-8.

KAYAL, Dhrubojyoti. *Pro Java EE Spring Patterns: best practices and design strategies implementing Java EE patterns with the spring framework*. Berkeley: Apress, 2008. 323 s. ISBN 978-1-4302-1009-2.

KEITH, Mike and Merrick SCHINCARIOL. *Pro JPA 2: A definitive guide to mastering the Java Persistence API*. 2nd ed. Berkeley: Apress, 2013. 484 s. ISBN 978-143-0249-269.

KUMAR, Santosh. *Spring and Hibernate*. 2nd ed. New Delhi: McGraw-Hill Education, 2013. ISBN 978-1-25-906372-5.

LAYKA, Vishal. *Learn Java for Web development*. New York: Apress, 2014. 447 s. ISBN 978-1-4302-5983-1.

LÜPPKEN, Sven a Markus STÄUBLE. *Spring Web Flow 2 web development: master Spring's well-designed web frameworks to develop powerful web applications*. Birmingham, U.K.: Packt Pub., 2009. 254 s. ISBN 978-1-847195-42-5.

MATHA, Mahesh P. *Jsp and servlets: a comprehensive study*. S.l.: Prentice-Hall Of India Pv, 2013. ISBN 978-812-0347-458.

PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky*. 2., aktualiz. a rozš. vyd. Praha: Grada, 2009. 570 s. ISBN 978-80-247-2653-3.

REESE, Richard M. a Jennifer L. REESE. *Java 7 new features cookbook: over 100 comprehensive recipes to get you up-to-speed with all the exciting new features of Java 7: quick answers to common problems*. Olton Birmingham: Packt Pub, 2012. 367 s. ISBN 978-1-84968-562-7.

RICO, D. F., H. H. SAYANI a S. SONE. *The business value of agile software methods: maximizing ROI with just-in-time processes and documentation*. Fort Lauderdale, FL: J. Ross Pub., 2009. 214 s. ISBN 978-1-60427-031-0.

SCARIONI, Carlo. *Pro spring security*. New York: Apress, 2013. ISBN 978-1-4302-4818-7.

SCHILD, Herbert. *Java the Complete Reference*. 9th ed. New York: McGraw-Hill Education, 2014. ISBN 978-0-07-180856-9.

SONATYPE Company Staff., (2014). *Maven: the Definitive Guide*. 2nd ed. Birmingham: Packt Pub., 2013. ISBN 978-144-9362-805.

STEPHENS, Rod. *Beginning Software Engineering*. Indianapolis: Wrox/Wiley Pub., 2015. ISBN 978-1-118-96914-4.

WALLS, Craig. *Spring in action*. 3rd ed. Shelter Island: Manning, 2011. 400 s. ISBN 978-1-935-18235-1.

YENER, Murat a Alex THEEDOM. *Professional Java EE Design Patterns*. Indianapolis: Wrox/Wiley Pub., 2015. 264 s. ISBN 978-1-118-84341-3.

Seznam zkratek

AJAX	–	Asynchronous JavaScript and XML
AUP	–	Agile Unified Process
CSRF	–	Cross-Site Request Forgery
CSS	–	Cascading Style Sheets
ČVOP	–	Číslo výpovědi opouštěného poskytovatele
CVS	–	Concurrent Versions System
DAO	–	Data Access Object
ERP	–	Enterprise resource planning
GCC	–	GNU Compiler Collection
HQL	–	Hibernate Query Language
HSQL	–	Hibernate Structured Query Language
HTTP	–	Hypertext Transfer Protocol
ICCID	–	Integrated Circuit Identity
IDE	–	Integrated Development Environment
IoC	–	Inversion of Control
JDBC	–	Java Database Connectivity
JSF	–	JavaServer Faces
JSR	–	Java Specification Requests
JSTL	–	JavaServer Pages Standard Tag Library
KPČ	–	Kód přenosu čísla
LDAP	–	Lightweight Directory Access Protocol
MVC	–	Model View Controller

PDF	–	Portable Document Format
POM	–	Project Object Model
PPS	–	Privátní podniková síť
RAC	–	Real Application Clusters
RMI	–	Remote Method Invocation
RUP	–	Rational Unified Process
SMTP	–	Simple Mail Transfer Protocol
SQL	–	Structured Query Language
SŘBD	–	Systém řízení báze dat
SVN	–	SubVersion
URL	–	Uniform Resource Locator
VPD	–	Virtual Private Database
WAR	–	Web application archive
XHTML	–	Extensible Hypertext Markup Language
XP	–	Extrémní programování
XSS	–	Cross-site scripting

Prohlášení o využití výsledků diplomové práce

Prohlašuji, že

- jsem byl seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;
- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3);
- souhlasím s tím, že diplomová práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že bibliografické údaje o diplomové práci budou zveřejněny v informačním systému VŠB-TUO;
- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;
- bylo sjednáno, že užít své dílo, diplomovou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 8. 7. 2015

.....

jméno a příjmení studenta

Seznam příloh

Příloha č. 1 – Diagram případů užití

Příloha č. 2 – Vybrané specifikace diagramu případů užití

Příloha č. 3 – Entity Relationship Diagram

Příloha č. 4 – Datový slovník

Příloha č. 5 – Konfigurační schéma beanů webových toků

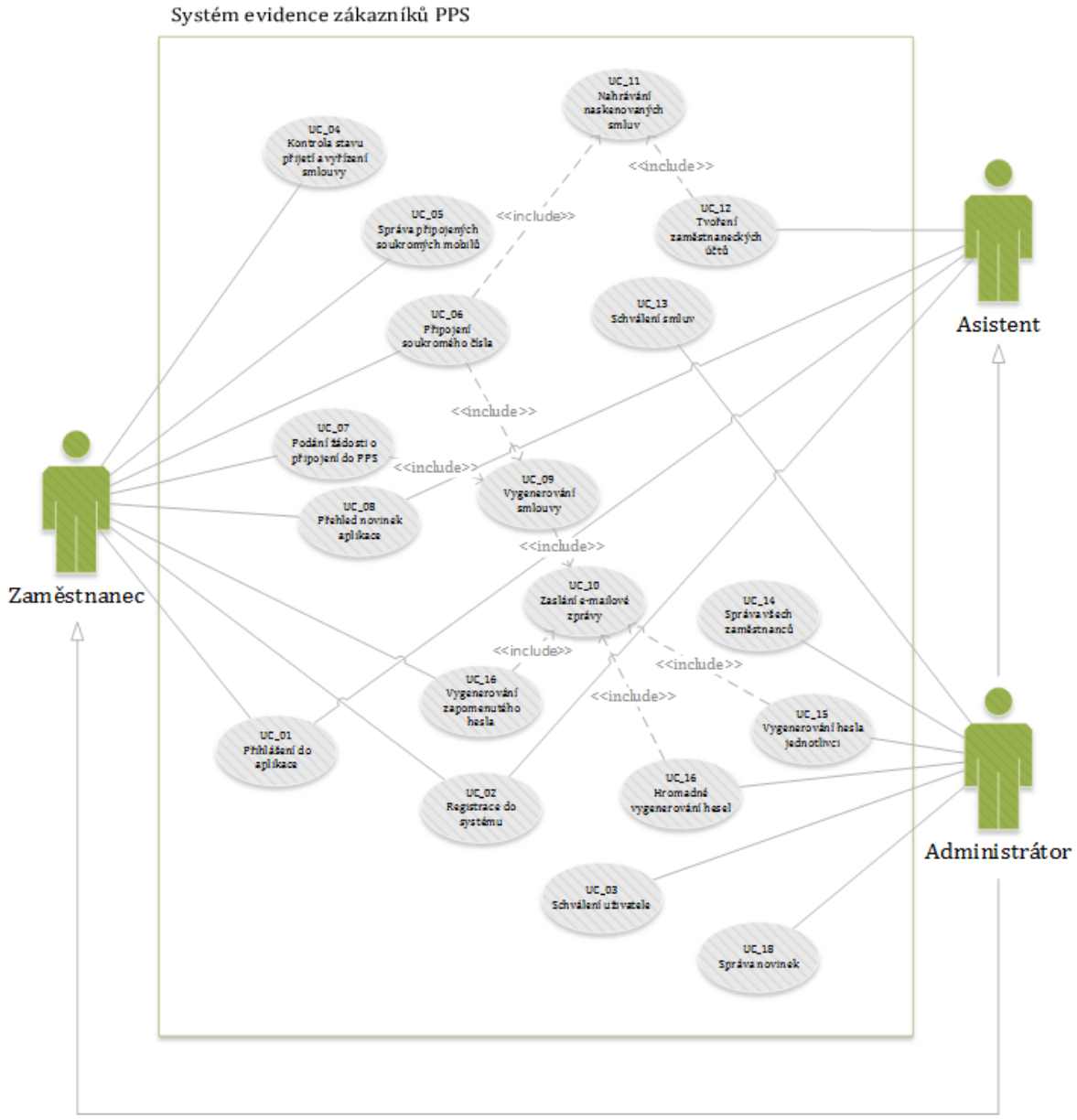
Příloha č. 6 – Diagram tříd pro správu uživatele aplikace

Příloha č. 7 – Ukázkový kód entitních tříd s dědičností

Příloha č. 8 – Náhledy vytvořené aplikace

Přílohy

Příloha č. 1 – Diagram případů užití



Příloha č. 2 – Vybrané specifikace diagramu případů užití

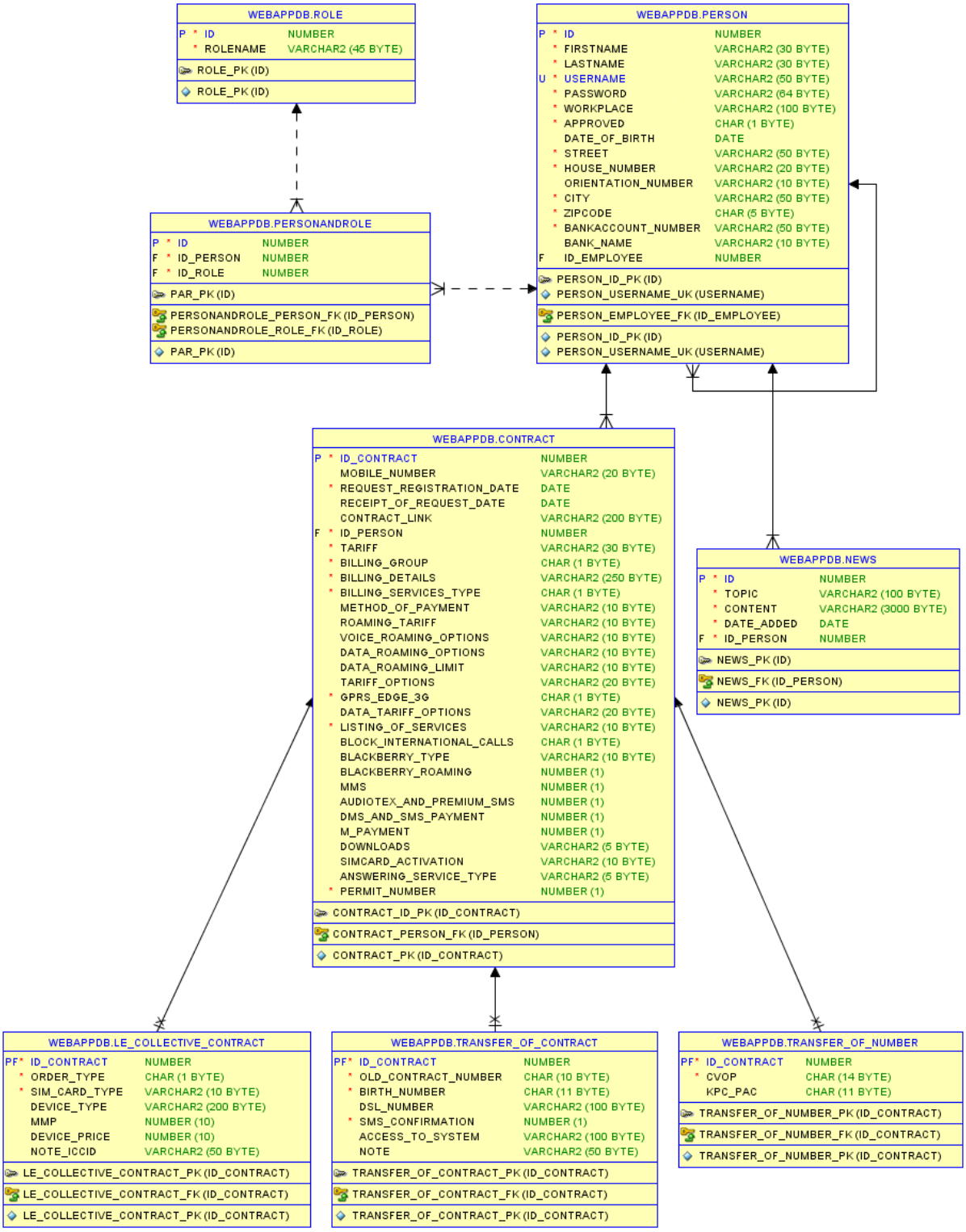
Případ užití: Registrace do systému
ID: UC_02
Účastníci: Zaměstnanec, Administrátor, Systém
Vstupní podmínky: -
Tok událostí: 1. Případ užití začíná po výběru položky *Nová registrace* 2. Systém zobrazí formulář pro registraci 3. Zaměstnanec vyplní údaje: jméno, příjmení, e-mail, město pracovního střediska a heslo 4. Zaměstnanec zvolí, že souhlasí s podmínkami o ochraně osobních údajů a zvolí dokončení registrace 5. KDYŽ Systém nevyhodnotí žádné chyby ze vstupních polí 1. Zaměstnanec je registrován jako nový uživatel, který čeká na schválení administrátorem 6. KDYŽ Administrátor uživatele ověří a schválí (pro schválení je použito UC_03 – Schválení uživatele) 1. Administrátor přidá zaměstnanci uživatelská práva 2. Zaměstnanec smí plně využívat aplikace
Následné podmínky: 1. Zaměstnanec má vytvořený účet jako uživatel

Případ užití: Připojení soukromého čísla
ID: UC_06
Účastníci: Zaměstnanec, Administrátor, Systém
Vstupní podmínky: 1. Zaměstnanec je přihlášený do systému 2. Nepřipojil do PPS více než 4 soukromé čísla
Tok událostí: 1. Případ užití začíná po výběru položky *Připojit soukromé číslo* 2. Systém zobrazí formulář pro připojení čísla 3. Zaměstnanec vybere typ smlouvy pro připojení soukromého čísla 4. Zaměstnanec vyplní údaje: jméno, příjmení, adresu, mobilní číslo, e-mail a číslo bankovního účtu 5. Systém použije UC_09 – Vygenerování smlouvy 6. Systém odešle vygenerovanou smlouvu na e-mail (viz UC_10 – Zaslání e-mailové zprávy) 7. Zaměstnanec smlouvu podepíše a vloží naskenovanou zpět do systému (viz UC_11 – Nahrávání naskenovaných smluv) 8. KDYŽ Administrátor schválí smlouvu (pro schválení je použito UC_13 – Schválení smluv) 1. Systém zavede soukromý číslo jako platné číslo PPS
Následné podmínky: 1. Zaměstnanec připojil soukromé číslo do PPS

Případ užití: Zaslání e-mailové zprávy
ID: UC_10
Účastníci: Systém
Vstupní podmínky: 1. Je zadán příjemce, předmět a zpráva e-mailové zprávy
Tok událostí: 1. Případ užití začíná jeho vyvoláním 2. Systém nastaví příslušný poštovní server, port a hlavičku e-mailu 3. Systém použije ze vstupu e-mailovou adresu příjemce, předmět a zprávu e-mailu 4. Systém přidá k e-mailové zprávě logo neziskové organizace 5. KDYŽ Systém nevyhodnotí žádné chyby z nastavených hodnot nebo při odesílání e-mailu 1. Systém zašle e-mailová zprávu příslušnému příjemci
Následné podmínky: 1. Systém úspěšně odeslal e-mailovou zprávu

Případ užití: Hromadné vygenerování hesel
ID: UC_16
Účastníci: Administrátor, Systém
Vstupní podmínky: 1. Administrátor je přihlášen do systému
Tok událostí: 1. Případ užití začíná po výběru položky z menu *Generovat hesla - Hromadně* 2. Systém zobrazí formulář s oznámením o vygenerování hesel všem zaměstnancům a položkou pro zadání hesla administrátora 3. Administrátor zadá své heslo 4. KDYŽ Systém nevyhodnotí žádné chyby ze vstupního pole 1. Systém vygeneruje všem uživatelům nová hesla 2. Systém zašle e-mailovou zprávu s novým heslem všem uživatelům (viz UC_10 – Zaslání e-mailové zprávy)
Následné podmínky: 1. Všem uživatelům zůstaly vygenerována nová hesla

Příloha č. 3 – Entity Relationship Diagram

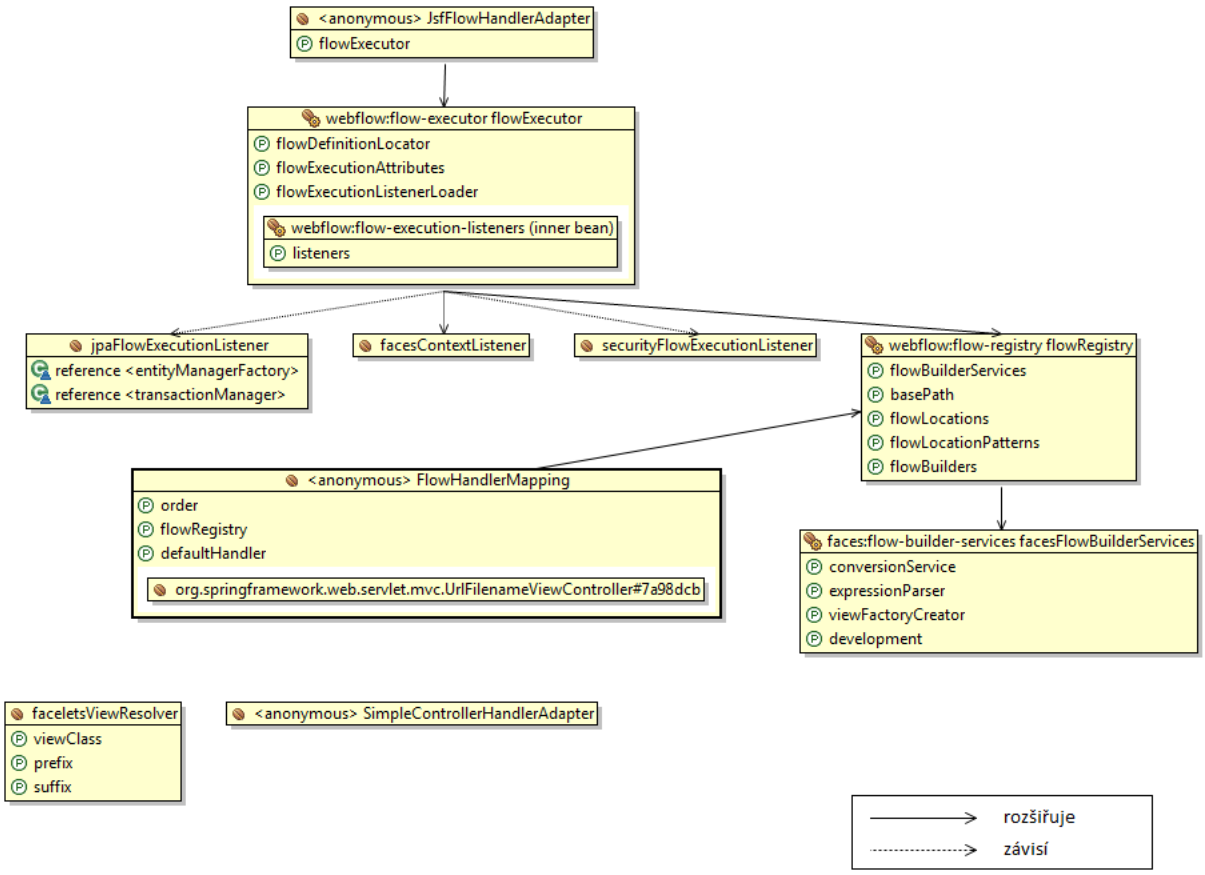


Příloha č. 4 – Datový slovník

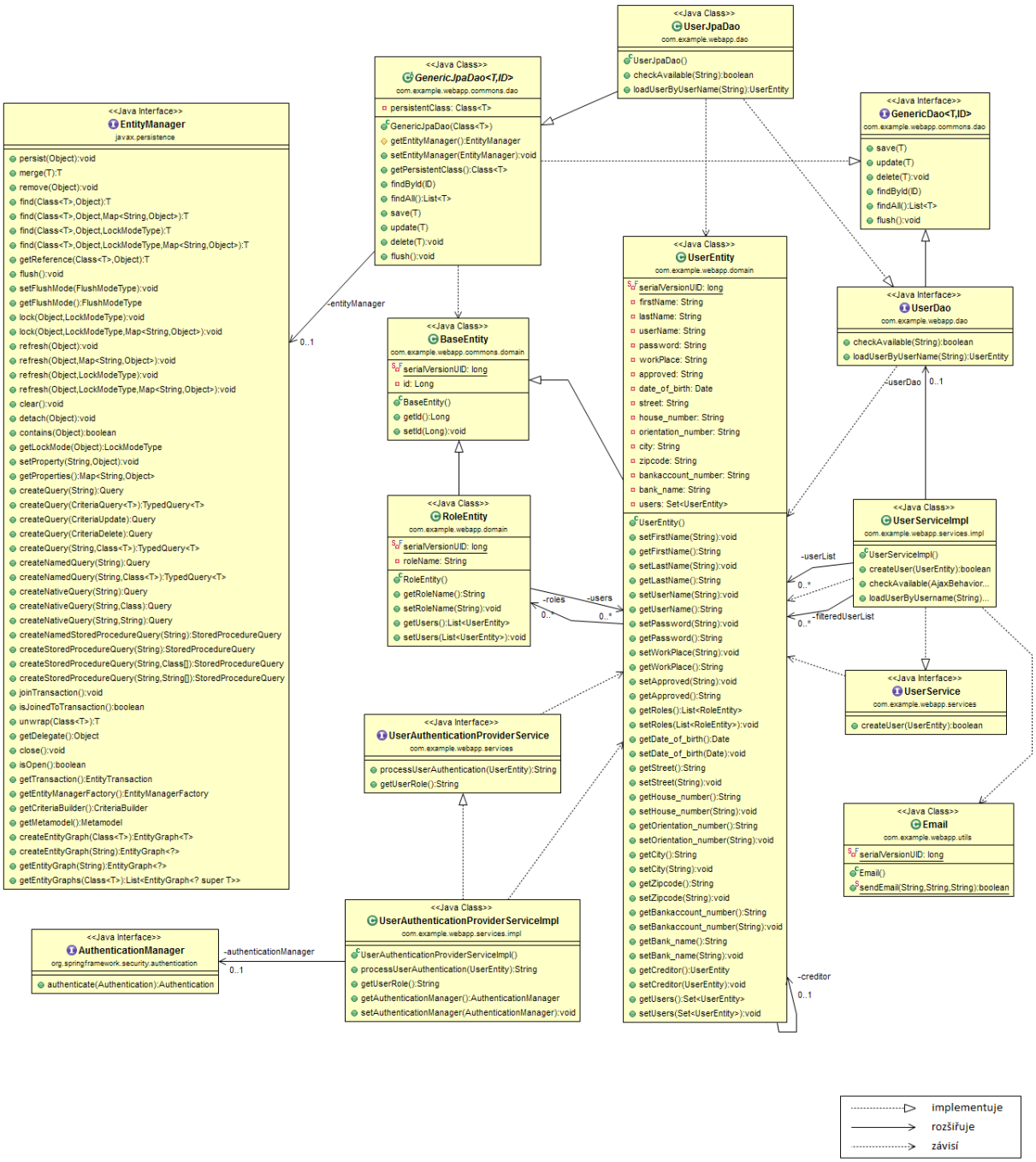
Název	Datový typ	Nulový	Popis
CONTRACT			Smlouva
ID_CONTRACT (PK)	NUMBER	Ne	Identifikační číslo smlouvy
MOBILE_NUMBER	VARCHAR2(20)	Ano	Mobilní číslo
REQUEST_REGISTRATION_DATE	DATE	Ne	Den zavedení žádosti
RECEIPT_OF_REQUEST_DATE	DATE	Ano	Den přijetí žádosti
CONTRACT_LINK	VARCHAR2(200)	Ano	Odkaz na smlouvu
ID_PERSON (FK)	NUMBER	Ne	Identifikační číslo osoby
TARIFF	VARCHAR2(30)	Ne	Tarif
BILLING_GROUP	CHAR(1)	Ne	Fakturační skupina
BILLING_DETAILS	VARCHAR2(250)	Ne	Podrobnosti fakturační skupiny
BILLING_SERVICES_TYPE	CHAR(1)	Ne	Typ vyúčtování služeb
METHOD_OF_PAYMENT,	VARCHAR2(10)	Ano	Způsob úhrady
ROAMING_TARIFF	VARCHAR2(10)	Ano	Roamingový tarif
VOICE_ROAMING_OPTIONS	VARCHAR2(10)	Ano	Hlasové roam. zvýhodnění
DATA_ROAMING_OPTIONS	VARCHAR2(10)	Ano	Datové roam. zvýhodnění
DATA_ROAMING_LIMIT	VARCHAR2(10)	Ano	Data roaming limit
TARIFF_OPTIONS	VARCHAR2(20)	Ano	Tarifní zvýhodnění
GPRS_EDGE_3G	CHAR(1)	Ne	GPRS, EDGE a 3G
DATA_TARIFF_OPTIONS	VARCHAR2(20)	Ano	Datové tarifní zvýhodnění
LISTING_OF_SERVICES	VARCHAR2(10)	Ne	Podrobný výpis služeb
BLOCK_INTERNATIONAL_CALLS	CHAR(1)	Ano	Blokování mez. hovorů
BLACKBERRY_TYPE	VARCHAR2(10)	Ano	Typ BlackBerry
BLACKBERRY_ROAMING	NUMBER(1)	Ano	BlackBerry roamingu
MMS	NUMBER(1)	Ano	multimediálních zpráv
AUDIOTEX_AND_PREMIUM_SMS	NUMBER(1)	Ano	Audiotexu a prémiové sms
DMS_AND_SMS_PAYMENT	NUMBER(1)	Ano	DMS a sms platby
M_PAYMENT	NUMBER(1)	Ano	M-platba
DOWNLOADS	VARCHAR2(5)	Ano	Stahování písní
SIMCARD_ACTIVATION	VARCHAR2(10)	Ano	Termín aktivace sim karty
ANSWERING_SERVICE_TYPE	VARCHAR2(5)	Ano	Typ záznamové služby
PERMIT_NUMBER	NUMBER(1)	Ne	Zařazení čísla do podnikové sítě
LE_COLLECTIVE_CONTRACT			Účastnická smlouva LE
ID_CONTRACT (PK, FK)	NUMBER	Ne	Identifikační číslo smlouvy
ORDER_TYPE	CHAR(1)	Ne	Typ objednávky
SIM_CARD_TYPE	VARCHAR2(10)	Ne	Typ sim karty
DEVICE_TYPE	VARCHAR2(200)	Ano	Typ zařízení
MMP	NUMBER(10)	Ano	Minimální měsíční plnění
DEVICE_PRICE	NUMBER(10)	Ano	Cena zařízení
NOTE_ICCID	VARCHAR2(50)	Ano	Poznámka s číslem ICCID
NEWS			Novinky
ID (PK)	NUMBER	Ne	Identifikační číslo novinky
TOPIC	VARCHAR2(100)	Ne	Nadpis
CONTENT	VARCHAR2(3000)	Ne	Obsah
DATE_ADDED	DATE	Ne	Datum přidání
ID_PERSON (FK)	NUMBER	Ne	Identifikační číslo osoby
PERSON			Osoba
ID (PK)	NUMBER	Ne	Identifikační číslo osoby

FIRSTNAME	VARCHAR2(30)	Ne	Jméno
LASTNAME	VARCHAR2(30)	Ne	Příjmení
USERNAME	VARCHAR2(50)	Ne	Uživatelské jméno (e-mail)
PASSWORD	VARCHAR2(64)	Ne	Heslo
WORKPLACE	VARCHAR2(100)	Ne	Město pracoviště
APPROVED	CHAR(1)	Ne	Schválen
DATE_OF_BIRTH	DATE	Ano	Datum narození
STREET	VARCHAR2(50)	Ne	Ulice
HOUSE_NUMBER	VARCHAR2(20)	Ne	Číslo popisné
ORIENTATION_NUMBER	VARCHAR2(10)	Ano	Číslo orientační
CITY	VARCHAR2(50)	Ne	Město
ZIPCODE	CHAR(5)	Ne	Poštovní směrovací číslo
BANKACCOUNT_NUMBER	VARCHAR2(50)	Ne	Číslo bankovního účtu
BANK_NAME	VARCHAR2(10)	Ano	Název banky
ID_EMPLOYEE (FK)	NUMBER	Ne	Identifikační číslo zaměstnance
PERSONANDROLE			Osoba a role
ID (PK)	NUMBER	Ne	Identifikační číslo osoby a role
ID_PERSON (FK)	NUMBER	Ne	Identifikační číslo osoby
ID_ROLE (FK)	NUMBER	Ne	Identifikační číslo role
ROLE			Role
ID (PK)	NUMBER	Ne	Identifikační číslo role
ROLENAME	VARCHAR2(45)	Ne	Název role
TRANSFER_OF_CONTRACT			Převod účast. smluv
ID_CONTRACT (PK, FK)	NUMBER	Ne	Identifikační číslo smlouvy
OLD_CONTRACT_NUMBER	CHAR(10)	Ne	Číslo zákaznické smlouvy
BIRTH_NUMBER	CHAR(11)	Ne	Rodné číslo
DSL_NUMBER	VARCHAR2(100)	Ano	DSL číslo
SMS_CONFIRMATION	NUMBER(1)	Ne	Sms potvrzení o převodu
ACCESS_TO_SYSTEM	VARCHAR2(100)	Ano	Přístup do T-Mobile systému
NOTE	VARCHAR2(50)	Ano	Poznámka
TRANSFER_OF_NUMBER			Přenos telefonního čísla
ID_CONTRACT (PK, FK)	NUMBER	Ne	Identifikační číslo smlouvy
CVOP	CHAR(14)	Ne	Číslo výpovědi
KPC_PAC	CHAR(11)	Ano	Číslo přenosu

Příloha č. 5 – Konfigurační schéma beanů webových toků



Příloha č. 6 – Diagram tříd pro správu uživatele aplikace



Příloha č. 7 – Ukázkový kód entitních tříd s dědičností

```
1.  @Entity
2.  @Table(name="contract")
3.  @Inheritance(strategy=InheritanceType.JOINED)
4.  public abstract class ContractEntity {
5.        @Id
6.        @Column(name="id_contract")
7.        @GeneratedValue
8.        private Long id;
9.        …
10. }
```

Kód pro mapování tabulky "Contract"

```
1.  @Entity
2.  @Table(name="le_collective_contract")
3.  @PrimaryKeyJoinColumn(name="id_contract")
4.  public class LeCollectiveContractEntity
                              extends ContractEntity {
5.        …
6.  }
```

Kód pro mapování tabulky "LE Collective Contract"

```
1.  @Entity
2.  @Table(name="transfer_of_contract")
3.  @PrimaryKeyJoinColumn(name="id_contract")
4.  public class TransferOfContractEntity
                              extends ContractEntity {
5.        …
6.  }
```

Kód pro mapování tabulky "Transfer of Contract"

```
1.  @Entity
2.  @Table(name="transfer_of_number")
3.  @PrimaryKeyJoinColumn(name="id_contract")
4.  public class TransferOfNumberEntity
                              extends ContractEntity {
5.        …
6.  }
```

Kód pro mapování tabulky "Transfer of Number"

Příloha č. 8 – Náhledy vytvořené aplikace